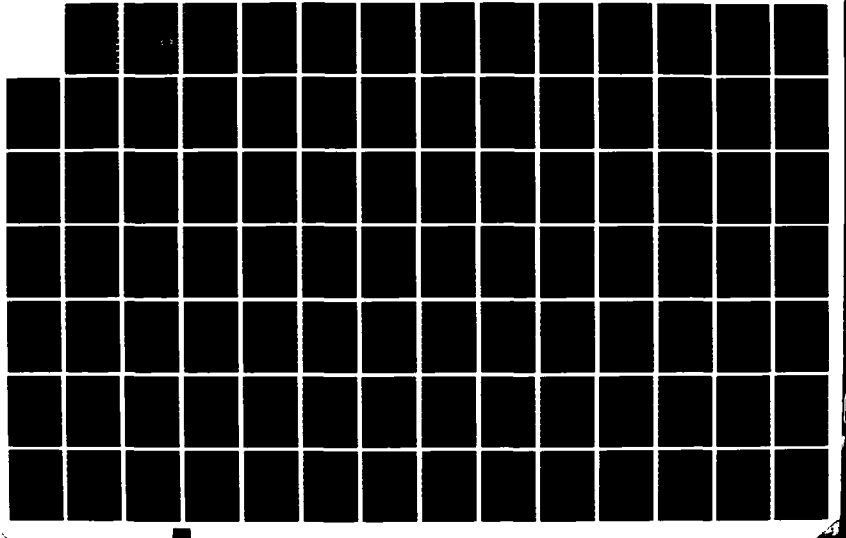
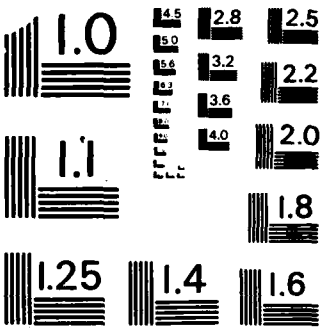
AD-A169 756 TOTALLY SELF-CHECKING CIRCUITS AND TESTABLE CMOS CIRCUITS(U) ILLINOIS UNIV AT URBANA COMPUTER SYSTEMS GROUP N K JHA JUN 86 CSG-51 N00039-80-C-0556 1/2

UNCLASSIFIED F/G 9/5 NL





June 1986

UILU-ENG-86-2217
CSG-51

(10)

COORDINATED SCIENCE LABORATORY
College of Engineering

TOTALLY SELF-CHECKING CIRCUITS AND TESTABLE CMOS CIRCUITS

Niraj K. Jha

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

unrestricted
SECURITY CLASSIFICATION OF THIS PAGE

AD-A169 756

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: none	1b. RESTRICTIVE MARKINGS: none
2a. SECURITY CLASSIFICATION AUTHORITY: not applicable	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE: not applicable	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): (CSG-51) UILU-ENG-86-2217	5. MONITORING ORGANIZATION REPORT NUMBER(S): not applicable
6a. NAME OF PERFORMING ORGANIZATION: University of Illinois Coordinated Science Laboratory; 6b. OFFICE SYMBOL (If applicable): n.a.	7a. NAME OF MONITORING ORGANIZATION: Office of Naval Research NESC; Semiconductor Research Corporation
6c. ADDRESS (City, State and ZIP Code): 1101 West Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State and ZIP Code): 2511 Jefferson Davis Hghwy. Arlington, VA 22202; P.O. Box 12053 Research Triangle Park, NC 27709
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: see 7a.; 8b. OFFICE SYMBOL (If applicable): n.a.	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00039-80-C-0556; SRC RSCH 84-06-049
8c. ADDRESS (City, State and ZIP Code): see 7b.	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
n.a.	n.a.	n.a.	n.a.

11. TITLE (Include Security Classification): Totally Self-Checking Circuits and Testable CMOS Circuits

12. PERSONAL AUTHOR(S): Niraj Kumar Jha

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Technical	FROM ____ TO ____	June 1986	116

16. SUPPLEMENTARY NOTATION: none

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): concurrent error detection, CMOS, Domino-CMOS, design, encoding, fault models, self-checking circuits, testability, nMOS

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A Totally Self-Checking (TSC) circuit belongs to a class of circuits used for Concurrent Error Detection (CED) purposes. It consists of a functional circuit that has encoded inputs and outputs and a checker that monitors these outputs and gives an error indication. It is known that the traditional stuck-at fault model is not sufficient to model realistic physical failures. Techniques for implementing existing gate-level TSC circuits in nMOS, Domino-CMOS and standard CMOS technologies, so that they are TSC with respect to physical failures, are described. Design methods which reduce the transistor count, delay, and the number of tests of TSC checkers are also given.

Another problem in the area of TSC circuits concerns embedded checkers whose inputs are not directly controllable. If they do not get all the required codewords to test them they cannot be guaranteed to be TSC. A new encoding technique and a design procedure to solve this problem are presented.

It has been shown previously that the two-pattern tests used to test CMOS circuits can be invalidated by timing skews. A necessary and sufficient condition is derived to find out whether or not an AND-OR or an OR-AND CMOS realization exists for a given function so that a valid test set can always be found, even in the presence of arbitrary timing skews. A new Hybrid CMOS realization is introduced to take care of the cases in which this is not possible. Keywords

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	unrestricted

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
		NONE

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

unrestricted
SECURITY CLASSIFICATION OF THIS PAGE

TOTALLY SELF-CHECKING CIRCUITS AND TESTABLE CMOS CIRCUITS

Niraj K. Jha

This work was supported in part by the Naval Electronics Systems Command under VHSIC contract N00039-80-C-0556 ad in part by the Semiconductor Research Corporation under contract SRC RSCH 83-01-014.

TOTALLY SELF-CHECKING CIRCUITS AND TESTABLE CMOS CIRCUITS

Niraj K. Jha
Computer Systems Group
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

ABSTRACT

A Totally Self-Checking (TSC) circuit belongs to a class of circuits used for Concurrent Error Detection (CED) purposes. It consists of a functional circuit that has encoded inputs and outputs and a checker that monitors these outputs and gives an error indication. It is known that the traditional stuck-at fault model is not sufficient to model realistic physical failures. Techniques for implementing existing gate-level TSC circuits in nMOS, Domino-CMOS and standard CMOS technologies, so that they are TSC with respect to physical failures, are described. Design methods which reduce the transistor count, delay and the number of tests of TSC checkers are also given.

Another problem in the area of TSC circuits concerns embedded checkers whose inputs are not directly controllable. If they do not get all the required codewords to test them they cannot be guaranteed to be TSC. A new encoding technique and a design procedure to solve this problem are presented.

It has been shown previously that the two-pattern tests used to test CMOS circuits can be invalidated by timing skews. A necessary and sufficient condition is derived to find out whether or not an AND-OR or an OR-AND CMOS realization exists for a given function so that a valid test set can always be found, even in the presence of arbitrary timing skews. A new Hybrid CMOS realization is introduced to take care of the cases in which this is not possible.

TOTALLY SELF-CHECKING CIRCUITS AND TESTABLE CMOS CIRCUITS

BY

NIRAJ KUMAR JHA

B.Tech., I.I.T., Kharagpur, India
M.S., S.U.N.Y. at Stonybrook

THESIS

submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Electrical and Computer Engineering in the Graduate College of the University of Illinois at Urbana-Champaign

Thesis Advisor: Professor Jacob A. Abraham

Urbana, Illinois

ACKNOWLEDGMENT

I would like to thank my thesis advisor, Professor Jacob A. Abraham, for his guidance and support through the research and writing of this thesis. I would also like to thank my colleagues in the Computer Systems Group for their helpful suggestions and advice. Thanks are also due to Ms. Jackie Ziemer and Ms. Marty North for their help throughout my stay here.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Rapid advances in the area of Very Large Scale Integrated (VLSI) circuits have allowed complex circuits to be integrated on a single chip. This explosive growth in the potential of integrated circuits (ICs), however, has its associated problems of testability and reliability. Several techniques have been proposed to solve these problems. The usefulness of these techniques varies according to the cost and speed considerations, and from application to application.

With the advent of cheap hardware, considerations of reliability and availability assume greater importance. Redundancy in hardware to meet these considerations is no longer limited to unattended environments, such as a space vehicle. A minimal system down time is a prerequisite for applications like telephone switching and industrial control. This is spurring interest in obtaining better designs and techniques for exploiting the power that VLSI puts at our disposal, and meeting the challenges posed by it.

The use of off-line [1] testing to check computers is becoming less appealing day-by-day because of the attendant down time, and also due to its failure to detect transient errors due to environmental causes like cosmic rays etc. Self-checking circuits [2] offer an attractive way of concurrently detecting transient and permanent errors with normal operation of the computer, with relatively little extra cost. Totally Self-Checking (TSC) circuits [3] are an important class of circuits which meet the Concurrent Error Detection (CED) requirement. The inputs to the TSC functional circuit are encoded and the circuit operates on these inputs to produce encoded outputs. TSC checkers are used to

monitor the outputs and provide an error indication when a non-code word is detected.

TSC circuits have some interesting properties. Assume a fault set in the circuit and that errors caused by the faults in the fault set are detected by the encoding. Its *self-testing* property guarantees that every fault in the fault set will be detected by at least one codeword. This property is required to flush out any latent faults present in the circuit. The *fault-secure* property guarantees that, until the fault is detected, all the outputs will be correct. A TSC checker detects faults by mapping non-code inputs to non-code outputs. This is referred to as the *code-disjoint* property.

The additional advantage of TSC circuits is that they do not use excessive redundancy such as required in schemes employing quadded logic [4], triple modular redundancy [5], and the more general n-tuple modular redundancy [6].

Since MOS is the dominant technology today, it is only logical to attempt to find efficient methods of implementing circuits in this technology. The physical failures that occur in the MOS technology, however, necessitate a fresh look at existing designs before their MOS implementation. The purpose of the research in this thesis was to investigate some problems associated with TSC circuits and their MOS implementations, and give a solution to the testability problem of CMOS circuits.

1.1. Literature Review

Since their introduction a little over a decade ago TSC circuits have found considerable use in fault-tolerant systems [7-15]. Research in this area has covered various designs like design of microprogram control units [7,8], adders [9], sequential machines [10], checkers [11-14], and combinational functional circuits [15]. Previous works investigating the problems of MOS implementations of TSC circuits were reported

in [16,17].

The stuck-at fault model has been shown to be inadequate for MOS technology [18]. Failures, such as shorts, are difficult to detect by a test set based on this fault model. Wadsack [19] has developed a "stuck-open" fault model to characterize failures in CMOS circuits. Presence of a stuck-open fault changes the behavior of the circuit from a combinational to a sequential one. Hence conventional testing approaches fail to detect such a fault. Additional problems with testability of CMOS circuits have come to light and have been discussed in [20].

1.2. Thesis Outline

In Chapter 2 we give the concepts behind TSC circuits, and the assumptions they work on. A realistic physical failure model for MOS technology is also presented.

In Chapter 3 we show how to implement TSC circuits in nMOS and Domino-CMOS technologies [21]. Our implementations are made TSC with respect to physical failures actually observed in the field. Many of the device and interconnect failures are shown to be mapped to stuck-at faults or stuck-open faults. Conditions and layout rules are given for the detection of those failures which cannot be so mapped, notably some diffusion and metal shorts; following the design rules allows those failures to be detected by the input codewords.

In Chapter 4 we investigate the implementation of TSC circuits in CMOS technology [22]. Tests generated under a static behavior assumption (zero delays assumed through all gates annd paths of the circuit) have been shown to be invalidated in the presence of timing skews in the variables changes and unequal delays in the circuit [20]. We have obtained a new CMOS realization whose use, in the case of TSC circuits,

guarantees the self-testing property even in the presence of arbitrary delays.

In Chapter 5 we present some techniques for reducing the transistor count of MOS TSC checkers [23]. These techniques include transfer of fanouts, removal of inverters, and use of multi-level realizations of functions. They also increase the speed of the circuit and may reduce the number of tests. Impressive reductions of up to 90% in the transistor count have been obtained in some cases. This directly translates into saving of chip area.

In Chapter 6 the problem associated with embedded checkers is addressed. These are checkers whose inputs are not directly accessible. The problem with such checkers is that they do not always get all the codewords from the functional circuit that are required to make them self-testing. A new encoding technique is presented in this chapter to encode the outputs of the functional circuit, so that the embedded checker based on this encoding is made self-testing by inputs received during normal operation [24]. This obviates the need for a direct control over the input lines of the embedded checker, in order to test it for faults.

It has been shown that there exist some functions for which CMOS realizations cannot be guaranteed to be testable in the presence of arbitrary delays in the circuit. In Chapter 7 we present necessary and sufficient conditions to find out *a priori* whether a valid test set exists for given CMOS realizations or not [25]. If such a test set does not exist then we solve the problem by introducing a new CMOS realization for which a valid test set is guaranteed.

CHAPTER 2

TOTALLY SELF-CHECKING CIRCUITS

2.1. Introduction

In this chapter we first discuss a realistic failure model that can be used to model failures observed in the field in MOS technology. Then we give the basic concepts behind TSC circuits. The assumptions under which these circuits work are also given.

2.2. Failure Model

Before providing the general physical failure model we briefly discuss the stuck-at fault model. This fault model assumes that most failures in digital circuits result in a constant logical value on the input or output lines.

A *single stuck-at fault* refers to the fault caused by a single line stuck-at 0 or 1. A *multiple stuck-at fault* refers to multiple stuck lines. If all lines are stuck to the same logical value in a multiple fault, it is referred to as a *unidirectional stuck-at fault.*

The effect of faults on the logical values of circuit outputs is denoted as an error. A *single error* refers to an error in which the logical value of only one circuit output line is changed. A *multiple error* refers to an error in which the logical values of multiple lines are changed. A *unidirectional error* refers to a multiple error in which all the logical values change in the same direction, either from 0 to 1 or 1 to 0, but not both.

With the increasing integration capability of MOS technology, the stuck-at fault model has been shown to be inadequate for modeling all realistic physical failures. Thus a better failure model is needed. Table 2.1 shows the device and interconnect failures that are observed in the field in the case of MOS technology. This table has been taken

from [26]. It will be the basis of the circuits that we develop later.

Table 2.1 A realistic physical failure model

Device failures	Interconnect failures
Gate-Drain short	Short between
Gate-Source short	diffusion lines,
Drain-contact open	Aluminum Polysilicon
Source-contact open	crossover broken,
Gate-Substrate short	short between
Floating gate	metal lines

2.3. Concepts and Definitions

A Totally Self-Checking (TSC) circuit has encoded inputs and outputs. During normal operation the circuit operates on the encoded inputs to produce encoded outputs. A fault is tested by at least one codeword, and the resultant non-code word at the outputs is detected by the checker. This is illustrated by the general structure of TSC circuits shown in Figure 2.1.

We will use the following definitions of TSC circuits given by Anderson [3].

Definition 2.1: A circuit is *fault-secure* if for every fault from a prescribed fault-set the circuit never produces an incorrect code output for code inputs.

Definition 2.2: A circuit is *self-testing* if for every fault from a prescribed fault-set the circuit produces a non-code output for at least one code input.

Definition 2.3: A circuit is *totally self-checking* if it is both fault-secure and self-testing.

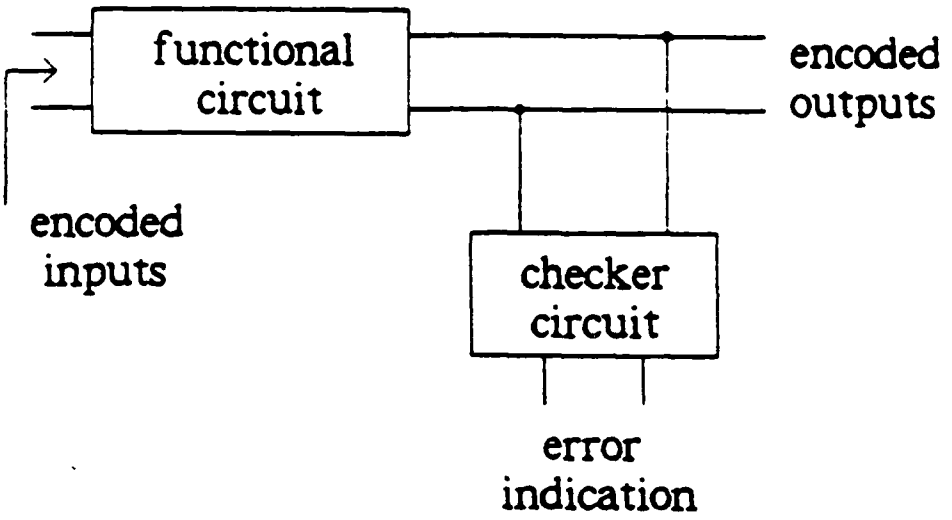


Figure 2.1 The TSC model

Definition 2.4: A circuit is *code-disjoint* if it always maps non-code inputs to non-code outputs.

Definition 2.5: A circuit is a *TSC checker* if it is both self-testing and code-disjoint.

The above definitions imply that an erroneous output caused by a fault in a fault-secure circuit is always a non-code word. Also, a self-testing circuit is diagnosable with just code inputs, and at least one code input causes a non-code output for every fault. If the first erroneous output from the circuit is a non-code word then the TSC goal is said to be achieved.

The TSC circuits thus have Concurrent Error Detection (CED) capability. Due to the on-line checking of faults the TSC circuit gives an error indication as soon as an error due to a permanent or transient fault is detected. Immediate detection of faults prevents the data from getting corrupted. Additionally, this obviates the need for having diagnostic software programs.

The assumptions under which the TSC circuits function are the following:

(1) Faults occur one at a time.

(2) The time interval between two faults is long enough so that all codewords can be applied to the circuit.

If an untested fault were to be allowed to be present in the TSC circuit, a subsequent fault could cause the circuit to lose its TSC property, which explains the assumptions. These assumptions are quite realistic because, generally, the input codewords are cycled through much faster than the Mean Time Between Failures (MTBF) of the circuit.

CHAPTER 3

TSC MOS CIRCUITS UNDER PHYSICAL FAILURES

3.1. Introduction

Existing realizations of TSC circuits are at the logic gate level, and assume a stuck-at fault model. As was mentioned earlier, all physical failures in MOS technology cannot be mapped to stuck-at faults. In [18] some general layout rules are given which either force a physical failure to be mapped to a stuck-at fault or prevent failures which do not map to stuck-at faults from occuring. Therefore a test set which detects stuck-at faults can still be used for an MOS implementation.

An electrical diagram of the MOS circuit cannot suggest all the physical failures that can occur in the circuit because it can be laid out in different ways. A possible short in one layout may not even be possible in another. Reports [27] and [28] take these considerations into account and give some layout rules to meet the TSC goal.

In this chapter we will look beyond an electrical diagram of the circuit, at its actual layout. TSC logic circuits designed at the logic gate-level will be implemented with complex MOS gates. This implementation, with the help of some layout rules, will be shown to be TSC with respect to realistic physical failures. The TSC MOS implementations given in this chapter were simulated in detail to ensure that the circuits were, indeed, TSC for physical failures, by using a MOS logic simulator MURPHY [26] written at the University of Illinois. This chapter will deal with nMOS and Domino-

CMOS TSC circuits, while the next chapter will be devoted to CMOS TSC circuits.

3.2. MOS implementation of TSC circuits

MOS technology enables us to have a single complex gate implement a function requiring at least two levels in its gate-level realization. A complex nMOS gate, for example, has a pull-up transistor and several control transistors. It acts like a switch network. By applying proper input patterns a set of conduction paths is established between the output node and ground. When one or more conduction paths are activated the output of the complex nMOS gate is logic 0, otherwise it is logic 1.

The integration capability of a complex MOS gate is limited by the following two constraints: (1) the maximum number of control transistors in any series path between the output node and ground, (AND fan-in) (2) the number of possible conduction paths between the output node and ground, (OR fan-in). Typically the upper limit for AND fan-in is four or five and for OR fan-in it is about fifty. The integration capability is also limited by the presence of: (1) inverting functions, and (2) fanouts.

We show below how logic functions can be modified so that they will require fewer transistors for their MOS implementations. In Chapter 5, techniques for reducing the complexity of the MOS implementations are discussed in detail.

Reducing the complexity of MOS implementations

Let a logic gate-level circuit realizing a function f be denoted as G_l(f) and its MOS implementation by G_m(f). To reduce the number of transistors required for the MOS implementation of G_l(f) one should first reduce the number of literals in the expression of f and obtain a modified $G_{l'}$(f) and its corresponding $G_{m'}$(f). Example 3.1 will make

things clear.

Example 3.1: Let $f = x_1.x_2+x_3.x_5+x_3.x_6+x_4.x_5+x_4.x_6$. Let $G_m(f)$ be an nMOS implementation consisting of a complex nMOS gate, comprising a pull-up transistor and ten control transistors, followed by an inverter. After reducing the literals, however, we get $f = x_1.x_2+(x_3+x_4).(x_5+x_6)$. Now $G_{m'}(f)$ requires only six control transistors in the complex nMOS gate.

Although $G_f(f)$ will generally have more levels than $G_l(f)$ it is important to note that $G_{m'}(f)$ is at least as fast as $G_m(f)$, as explained below.

The speed of a MOS implementation is roughly governed by the following factors:

(1) Maximum number of MOS gates any signal comes across, from circuit inputs to outputs.

(2) The AND fan-in of the constituent complex MOS gates.

(3) The OR fan-in of the constituent complex MOS gates.

$G_{m'}(f)$ is no different than $G_m(f)$ as far as these factors are concerned. Hence there is no speed degradation.

The above step is better from the self-testing point of view too. Since the above step reduces the number of transistors, the codewords required to test the transistors that have been removed are no longer required to ensure the self-testing property. So implementing $G_{m'}(f)$ instead of $G_m(f)$ saves on number of transistors and number of codewords required to test it.

We now switch our attention to detection of stuck-at faults at the inputs of transistors in any given $G_m(f)$. Apart from the input lines of inverters, every other input

line in $G_m(f)$ has a corresponding line in $G_l(f)$ realizing the same function. So a codeword which detects a stuck-at fault on a given line in $G_l(f)$ also detects the stuck-at fault at the input of the corresponding transistor in $G_m(f)$. Since the stuck-at faults at the outputs of all inverters in $G_m(f)$ are detectable by the above argument, it trivially follows that stuck-at faults on the input lines of these inverters will also be detectable. So $G_l(f)$ and $G_m(f)$ require the same test set to make them self-testing with respect to stuck-at faults. The same is obviously true for $G_f(f)$ and $G_{m'}(f)$.

3.3. Failure set

We will use the failure model given in Table 2.1. In our failure-set we include all the physical failures mentioned in this table. Shorts are allowed among any number of adjacent metal lines which feed the same complex MOS gate, or any two adjacent metal lines feeding different complex MOS gates. Horizontal or near-horizontal shorts among any number of adjacent diffusion lines internal to a complex MOS gate are allowed, as also any diffusion short between any two complex MOS gates. Metal lines can be run on one or more levels.

If arbitrary shorts were to be allowed among metal or among diffusion lines, then it would be quite impossible to guarantee a MOS implementation to be TSC with respect to these failures. Therefore, we follow a design methodology where inputs to a MOS gate are fed on horizontal metal lines and different conduction paths in the MOS gate from the output node to ground are realized by vertically running diffusion lines. We now show how MOS implementations of TSC circuits using this methodology can be made

TSC with respect to physical failures.

3.4. nMOS implementation of TSC circuits

In this section we present techniques to make an nMOS implementation TSC with respect to physical failures. In the next section we extend our techniques to Domino-CMOS technology.

We will consider the device failures first and then the interconnect failures. We ensure the validity of using the logic gate level stuck-at fault test set for testing the physical failures in its nMOS implementation by presenting some layout rules.

3.4.1. Self-testing under device failures

All the device failures, except gate-to-drain shorts, map to a stuck-at 0 (s-a-0) fault because they prevent the transistor from conducting and hence open the conduction path. The gate-to-drain short of a transistor cannot be modeled as a s-a-0 fault because the potential at the output node may assume an intermediate value which might be interpreted as a logic 0 or 1 depending on the environment of the complex nMOS gate. But one can verify that the stuck-at 1 (s-a-1) fault test for that transistor's input detects this failure even though this short cannot be modelled as a s-a-1 fault.

3.4.2. Self-testing under interconnect failures

3.4.2.1. Metal line short failures

Metal line shorts cannot, in general, be modelled as stuck-at faults but yet can be detected by stuck-at fault tests, if certain conditions are met. When two metal lines carrying inputs to a complex nMOS gate are shorted, their logical values are ANDed.

Let us first consider a short between two adjacent lines. We refer to a short between two adjacent lines as a 2-tuple short.

Conditions and layout rules for detection of metal line shorts

A 2-tuple short between metal lines feeding the same complex nMOS gate is detectable by a particular input vector if the following conditions are met:

(1) The output of the complex nMOS gate for this vector is logic 0.

(2) The vector causes logic 0 to appear on one line and logic 1 on the other.

(3) If we were to open the transistor to which logic 1 is fed, then the output of the complex nMOS gate becomes logic 1.

(4) The effect of Condition (3) above is observable at the circuit outputs.

For detection of a 2-tuple short between metal lines feeding different complex nMOS gates the Conditions (2) and (4) above remain the same, but Conditions (1) and (3) get slightly changed as follows:

(1)′ The output of at least one of the two complex nMOS gates is logic 0.

(3)′ If we were to open the transistor to which logic 1 is fed, then the output of the complex gate to which that transistor belongs should change from logic 0 to logic 1.

Now let us graduate to n-tuple shorts among adjacent metal lines, where $n>2$. Some n-tuple metal shorts among adjacent metal lines feeding different complex nMOS gates can cause the circuit to lose its fault-secure property. They are not included in our failure set based on the justification that such shorts are highly unlikely. For an n-tuple metal short among metal lines feeding the same nMOS gate the previously stated Conditions (2) and (3) for 2-tuple metal shorts are modified as follows:

(2)'' The input vector has a logic 0 on at least one of the n metal lines under consideration.

(3)'' If we were to open the transistors to which logic 1 is fed, then the output of the complex nMOS gate becomes logic 1.

Theorem 3.1: If an (n-1)-tuple metal short x_1-x_2-...-x_{n-1} is detected by a vector set $V_{12..(n-1)}$ and another (n-1)-tuple short x_2-x_3-...-x_n is detected by a vector set $V_{23..n}$ then the n-tuple metal short x_1-x_2-...-x_n is detected by those elements of the vector set $V_{12..n} = V_{12..(n-1)} \cup V_{23..n}$, for which Condition (4) is still satisfied.

Proof: It is easy to see that the vectors in the set $V_{12..n}$ satisfy the modified Condition (2)'' because elements of the sets $V_{12..(n-1)}$ and $V_{23..n}$ satisfy Condition (2)'' (Condition (2) for n=3). So when one of the vectors from $V_{12..n}$ is applied to the complex nMOS gate, x_1, x_2,.., x_n will all assume the value logic 0 due to the ANDing property. Since Condition (3)'' (Condition (3) for n=3) is satisfied by elements of the set $V_{12..(n-1)}$ and $V_{23..n}$, Condition (3)'' will also be satisfied by the elements of $V_{12..n}$, because having an additional logic 0 on one of the lines comprising the n-tuple short can only open more conduction paths. If Condition (4) is still satisfied, it means that the effect of the n-tuple short will be observable at the circuit outputs.

□

The useful result from the Theorem 3.1 is that if any 2-tuple constituent of an n-tuple (n>2) metal short is detectable then, in general, the n-tuple short is also detectable.

To ensure detection of metal shorts the following layout rules should be followed:

Rule 3.1: Choose a convenient configuration for the metal lines. If a possible 2-tuple metal short can exist which does not satisfy the above four conditions for any input vector, then exchange either one or both these metal lines with other metal lines feeding the same complex nMOS gate so that the conditions are satisfied. If this fails simply lay out these two lines sufficiently far apart to prevent a short.

In most self-checking circuits, however, the 2-tuple shorts are found to be detectable quite easily. Alternatively, if two-level metal lines are allowed, then one of the offending metal lines could be placed on another level thus precluding the possibility of the above short.

In [28] the problem of laying out power lines is considered. We will employ Rule 3.2 to prevent redundant shorts between power lines from occurring.

Rule 3.2: Do not layout power lines of the same type adjacently.

One can take advantage of a two-level metal implementation to ensure this.

Example 3.2: Let us consider the 3-out-of-6 checker design by Smith [15]. Figure 3.1(a) shows a possible way in which part of its nMOS implementation can be laid out. Table 3.1 below gives the codewords which detect the different 2-tuple metal shorts in Figure 3.1(a).

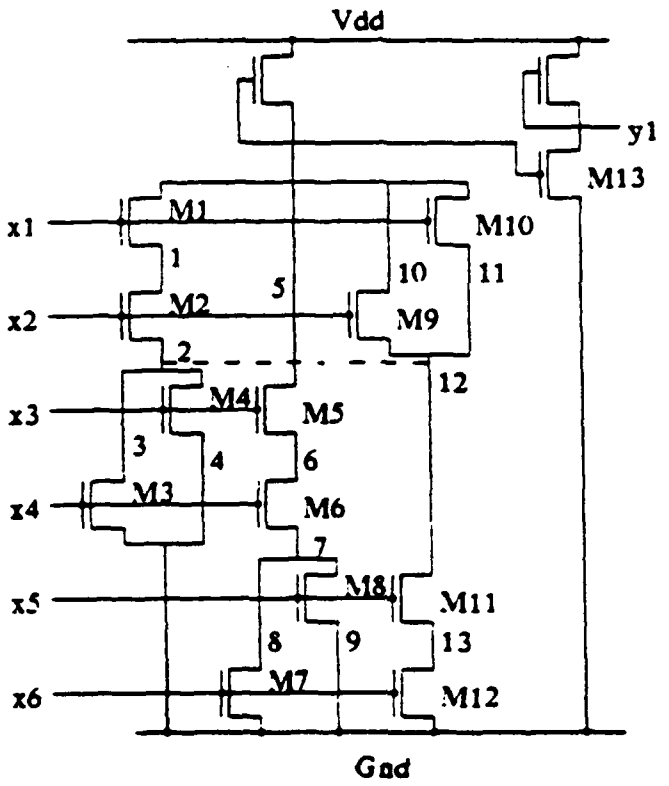


Figure 3.1(a). Part of the nMOS implementation of Smith's 3-out-of-6 code checker

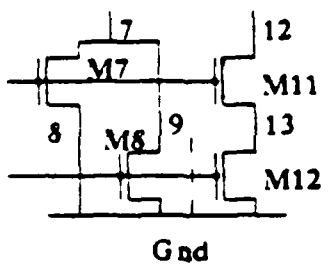


Figure 3.1(b). A different implementation of bottom portion of Figure 3.1(a)

Table 3.1. Detection of 2-tuple metal shorts

2-tuple metal short failure	vectors that detect it
x1-x2	010011, 100011
x2-x3	110100, 001110, 001101, 010011
x3-x4	110100, 111000
x4-x5	110100, 001101, 100011, 010011
x5-x6	001101, 001110

3.4.2.2. Diffusion short failures

First we will consider horizontal or near-horizontal shorts among any two adjacent vertically running diffusion lines in a complex nMOS gate. Most of the 2-tuple diffusion shorts can be found, by inspection, to be equivalent to s-a-1 faults at the inputs of different transistors.

Example 3.3: Considering Figure 3.1(a) again, Table 3.2 can easily be obtained by inspection to show the mapping of 2-tuple diffusion shorts to s-a-1 faults.

Table 3.2. Mapping of 2-tuple diffusion shorts to stuck-at faults

2-tuple diffusion short	s-a-1 fault at gate of transistor
1-5	M1
2-5	M1 or M2
3-4	M4
4-6	M6, M7 or M8
4-7	M7 or M8
5-12	M9 or M10
8-9	M8
10-11	M10
9-13	M12

These s-a-1 faults are known to be detectable, as explained in Section 3.2. Hence so are these 2-tuple diffusion shorts. In fact the only two possible 2-tuple shorts that were not listed above are 6-12 and 7-12. If we look carefully, the short 7-12 is found to be equivalent to a s-a-1 fault at the gate of M11 or M12. The short 6-12 will be found to be detectable because Conditions (1) and (2) given below are satisfied.

Conditions and layout rules for detection of diffusion short failures

The following two conditions have been given in [18] for detection of a 2-tuple short between nodes i and j:

(1) Activation of at least one conduction path between the output node and the node i (respectively j) and at least one conduction path between the node j (respectively i) and the ground node.

(2) Blocking of all other conduction paths of the network.

These two conditions ensure that the diffusion short will provide a path for the output node to discharge to logic 0, and will hence be detectable.

Next let us consider n-tuple (n>2) diffusion shorts internal to an nMOS gate. An example of a 3-tuple short is shown in Figure 3.1(a) with a dotted line.

Theorem 3.2: If any 2-tuple constituent of an n-tuple diffusion short internal to a complex nMOS gate is detectable then so is the n-tuple short.

Proof: The only way a 2-tuple diffusion short can be detectable is if without this short the output node of the complex gate is at logic 1 for any particular vector, and with the short it is pulled down to logic 0. Since the presence of the other constituents of the n-tuple short only creates additional conduction paths, it can only help in pulling down the output node to logic 0. So it is sufficient to have only one 2-tuple constituent detectable.

□

For example, in Figure 3.1(a), the two 2-tuple constituents of the 3-tuple short 2-5-12 are 2-5 and 5-12. Both are detectable and hence so is 2-5-12.

We will now give some layout rules so that some problems associated with diffusion short failures can be prevented.

Rule 3.3: If the same input is fed to two transistors in adjacent vertical conduction paths and if their drains are connected directly to the output node or if their sources are connected directly to the ground node then the following Rule (a) should be applied; if this is impossible, Rule (b) should be applied.

(a) Another vertical conduction path which does not have a transistor with the same input should be interposed between the two conduction paths.

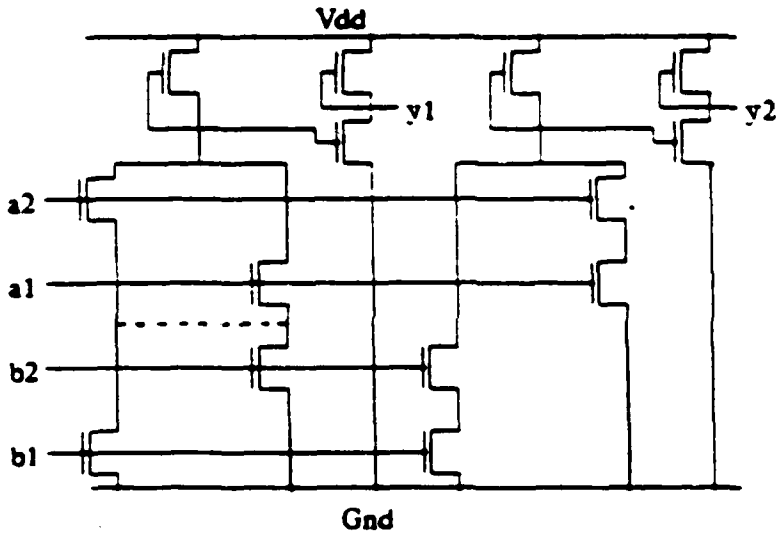
(b) A vertical diffusion line connected to the output node or the ground node (as the case may be) should be interposed between the two conduction paths.

Example 3.4: Let us first give an example illustrating Rule 3.3(a). If the bottom portion of Smith's 3-out-of-6 checker had been laid out as in Figure 3.1(b) then the 2-tuple diffusion short 9-13 would not be detectable. In fact, this short would make the circuit redundant. If a convenient conduction path had not been available for interposition then we could apply Rule 3.3(b). This is illustrated by the vertical dotted line in the Figure 3.1(b). Now the short 9-13 actually results in shorting these nodes to ground. This makes it equivalent to a s-a-1 fault at the input of transistor M8 or M12.

Rule 3.4: If a 2-tuple diffusion short is not found to be equivalent to a s-a-1 fault and if an input vector exists at the input of the complex nMOS gate such that for a particular configuration of transistors the above two Conditions (1) and (2) are satisfied then this configuration should be preferred over others.

Example 3.5: In [27] an nMOS implementation of a two-rail code checker is given with an undetectable 2-tuple diffusion short. This is reproduced in Figure 3.2, with the undetectable short given as a dotted line. We have also given the inputs that can be present for this two-rail code checker. The reason that the given short is undetectable is that the two conditions given above are not met for any of the four input vectors. Now consider the configuration of Figure 3.3; here, the same short is detectable because the two conditions are met.

Rule 3.5: If an internal horizontal diffusion short in any complex nMOS gate is not equivalent to any s-a-1 fault and Rules 3.3 and 3.4 do not alleviate the problem, then



a1	b1	a2	b2
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0

Figure 3.2 nMOS implementation of a two-rail code checker

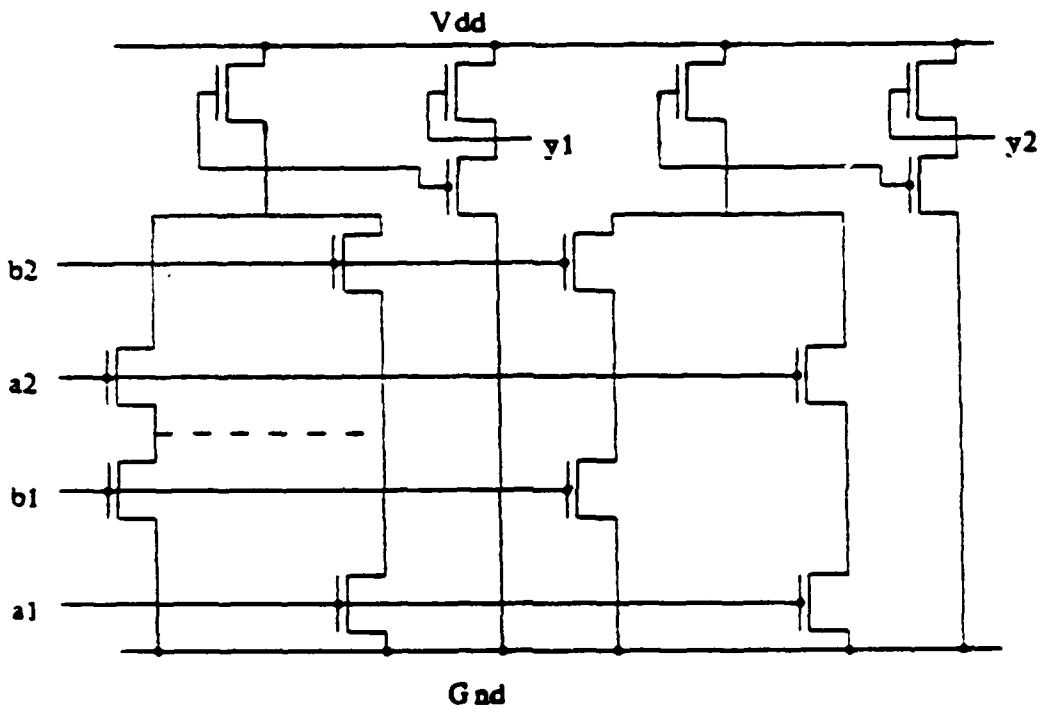


Figure 3.3 TSC nMOS implementation of a two-rail code checker

this short is prevented from occurring by placing the two diffusion lines sufficiently far apart.

Rule 3.6: Every complex nMOS gate is enclosed in a diffusion box and this box is connected to ground.

Rule 3.6 forces diffusion shorts between any two complex nMOS gates to be actually shorted to ground. So just as with Rule 3.3(b) these shorts can be modelled as s-a-1 faults.

3.4.2.3. Al-Poly crossover broken failure

If a metal line breaks due to an Al-Poly crossover broken failure then the *transistors whose gates were being fed* by that metal line will not be controllable. So the effect will be same as that of a floating gate failure which has been shown to be detectable. One might point out that more than one transistor can have a floating gate. But since floating gate failures are, in the static case, equivalent to s-a-0 faults this will open more conduction paths and can only aid in testing.

If the Aluminum line happens to be a power or ground line then the above failure might cause a unidirectional error. This has to be avoided in case of a TSC circuit whose output code-space is meant to detect single errors only. For this case [28] gives a layout rule which requires the complex MOS gates feeding different circuit outputs to be powered by different power lines.

Now we go on to investigate the fault-secure property under physical failures.

3.4.3. Fault-secure property

The fault-secure circuit always produces the correct code or a non-code in presence of failures, never an incorrect code. We show below how this property is ensured in presence of physical failures.

(1) *Under device failures:* All the device failures are internal to a complex nMOS gate. So for a given vector a 0-to-1 or 1-to-0 error can occur at the output node of such a complex nMOS gate. A suitable stuck-at fault on the corresponding line in the logic gate-level circuit is known not to produce an incorrect code at the circuit outputs, for this vector. Hence the fault-secure property of the nMOS implementation follows.

(2) *Under metal short failures:* If the shorted metal lines feed the same complex nMOS gate or do not have fanouts to other complex nMOS gates when they feed two different nMOS gates, then the arguments given in (1) above hold. But if one or both shorted metal lines have fanouts feeding more than two complex nMOS gates, then in case of TSC circuits whose code space detects single errors only, these two metal lines should be placed sufficiently apart, in order to avoid losing the fault-secure property.

(3) *Under diffusion short failures:* If the diffusion short is internal to a complex nMOS gate the arguments given in (1) hold again. Since the intergate diffusion shorts can be mapped to s-a-1 faults by using Rule 3.6, the fault-secure property will be maintained for such shorts too.

(4) *Under Al-Poly crossover broken failures:* If only one complex nMOS gate is affected then the arguments in (1) are valid again. But if this failure affects more than one complex nMOS gate then there may be a unidirectional error at the circuit outputs. If the code for the TSC gate-level design detects such errors then the nMOS implementation will be fault-secure with respect to this failure. If it catches single errors only then this problem can be avoided by rerouting the Polysilicon line so that only one complex nMOS gate can be affected.

Lastly, we now have to investigate the code-disjoint property.

3.4.4. Code-disjoint property

The code-disjoint property implies mapping of non-code inputs to non-code outputs. This property is required of all TSC checkers so that they indicate error as soon as it occurs. Since the gate level and the nMOS circuits both realize the same function, a non-code at the input of either will produce the same non-code at their outputs. Hence the nMOS implementation will also be code-disjoint.

3.5. Domino CMOS implementation of TSC circuits

CMOS technology is rapidly becoming the dominant MOS technology. Although standard CMOS circuits have the advantage that they require much less power than an equivalent nMOS circuit, they do pose a difficulty as far as implementation of self-checking circuits is concerned. This is due to the stuck-open faults [19] that can be caused by the device failures. A two-pattern test, consisting of an initialization input and a test input, is required to detect such faults. Even if each required two-pattern test was assumed to be available, under normal operation, they can still be invalidated by

timing skews as mentioned in [20].

Furthermore, standard CMOS has the disadvantage of requiring the realization of the same logic with both nMOS and pMOS networks and thus it wastes area. Lastly, the diffusion (active) shorts can cause both nMOS and pMOS networks to conduct at the same time. This fault is analog in nature because the output node assumes a voltage value somewhere between V_{dd} and ground depending on the impedance ratio of the paths from V_{dd} to the output node, and the output node to ground. This further complicates the problem of making the circuit self-testing.

The Domino-CMOS technique, developed by Krambeck et al [29], does not have most of the problems mentioned above for standard CMOS. It has a further advantage that it is compatible with standard CMOS.

Figure 3.4 shows the Domino-CMOS implementation of part of the two-rail code checker. The 'clock=0' phase is called the 'precharge phase' during which the output node is precharged high while the path to ground is opened. The 'clock=1' phase is called the 'evaluation phase' during which the output node is pulled low or remains high depending on whether or not a conduction path is established through the nMOS network.

Although device failures can still result in 'stuck-open' faults, fortunately the combination of the following two factors makes these detectable:

(1) precharging of the output node to logic 1.

(2) presence of a single series pMOS path from V_{dd} to output node. actually consisting of only one clocked pMOS transistor.

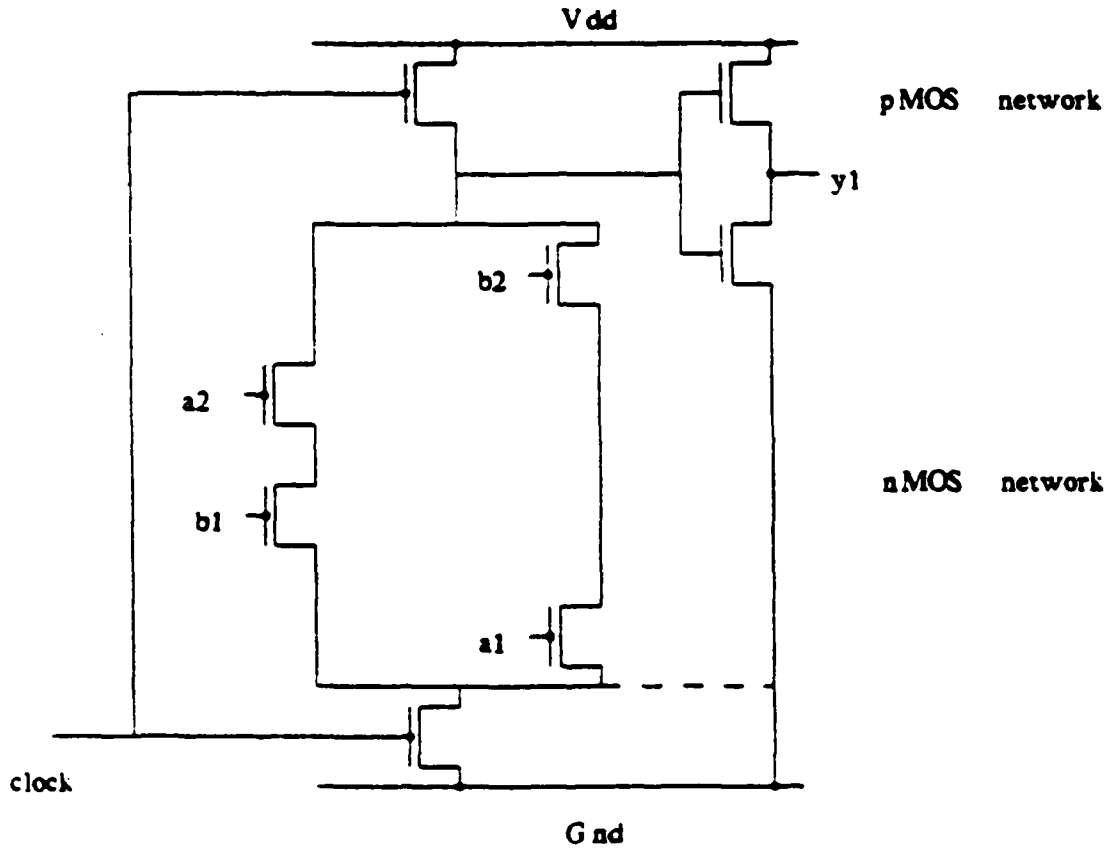


Figure 3.4 Domino-CMOS implementation of part of a two-rail code checker

The first factor obviates the need for initializing the output node of a complex Domino-CMOS gate to logic 1 to detect the stuck-open faults in the nMOS network. The second factor helps detect the stuck-open fault in the pMOS transistor. All the six device failures listed in Table 2.1 open the path from V_{dd} to the output node and are hence equivalent to a 'stuck-open' fault. These device failures in the clocked pMOS transistor cause the output node to eventually discharge to logic 0 (if it has not already been pulled down to logic 0 through the nMOS network). Hence they are detectable by a test which tests the output node for a s-a-0 fault (s-a-1 fault at the inverter output).

If any of the six device failures occur in the clocked nMOS transistor, it will result in any path from the output node to ground never getting activated. This is obviously detectable by a test for a s-a-1 fault at the output node.

The device failures in the pMOS transistor of the inverter are detected by a s-a-0 fault test at the inverter output. The device failures in the nMOS transistor of the inverter, however, require a s-a-0 fault test at the inverter output to be followed by a s-a-1 fault test [30].

We will now shift attention to interconnect failures in the Domino-CMOS implementation. The nMOS network of this implementation can be treated exactly as in Section 3.4. The only exception occurs when a diffusion (active) short as shown by the dotted line in Figure 3.4 takes place. This short is undetectable. To prevent this from happening the following layout rule should be observed:

Rule 3.7: The connection from the 'source' of the nMOS transistor of the inverter to ground is laid out in metal.

The arguments on fault-secure and code-disjoint properties given in Section 3.4 for nMOS circuits can now be seen to be trivially extensible to Domino-CMOS circuits as well.

CHAPTER 4

TSC CMOS CIRCUITS

4.1. Introduction

CMOS has become a popular technology for implementing LSI/VLSI circuits because of its low power requirement. In Chapter 3 we briefly mentioned the stuck-open fault model and the two-pattern tests that are required to detect these faults. Let us elaborate on this a little. The two-pattern tests consist of an initialization input and a test input. The initialization input initializes the output node of the CMOS gate to a logic value opposite to the value obtained when a test input is applied. The test input fails to establish a conduction path from the output node to V_{dd} (Gnd) if the stuck-open fault being tested is present in the pMOS (nMOS) network of the CMOS gate. This forces the CMOS gate to retain its previous logic value at its output node and the fault is detected. Such a two-pattern test can be found for all the stuck-open faults in an irredundant CMOS circuit.

Various methods have been proposed to detect stuck-open faults in CMOS combinational circuits [31-35]. Most of these methods generate tests based on static behavior of the CMOS circuits; in other words, they assume a zero delay through all gates and interconnections. It was, however, shown in [20] that these tests can be invalidated in the presence of arbitrary delays.

Another problem with CMOS circuits is that the stuck-on faults at the gates of the transistors cannot be guaranteed to be detectable by monitoring logic levels [32]. A solution that has been proposed is to monitor the steady state current drawn by the circuit when the required test is applied [36].

In [17] the problem of designing TSC CMOS circuits, which have valid tests in presence of arbitrary delays, has been addressed. But the authors have limited themselves to TSC checkers and do not consider TSC functional circuits. Furthermore, they consider checkers based on specific m-out-of-2m and two-rail codes only. Our results are more general and are applicable to any TSC CMOS circuit. We introduce a new CMOS realization, which we call a *Hybrid CMOS realization*, to implement the TSC circuits in CMOS technology. We will show that a valid test set is guaranteed to exist for a Hybrid CMOS realization, even in the presence of arbitrary delays, if a certain condition is met.

In this chapter we will concern ourselves with obtaining CMOS circuits which are TSC with respect to stuck-open and stuck-on faults only. It can be verified that most of the physical failures map to stuck-open and stuck-on faults in CMOS technology. Those which do not can be prevented from occurring by employing simple layout rules similar to those given in Chapter 3. In this way we can obtain CMOS circuits which are TSC with respect to realistic physical failures as well.

4.2. CMOS realizations

CMOS gates can be either primitive (i.e., AND, OR, NAND, NOR, NOT gates) or complex. A complex gate realizes those functions which require at least two levels in their gate-level realization. We will first give two CMOS realizations on which our newly proposed Hybrid CMOS realization is based. Many other general CMOS realizations are, of course, also possible; but we will only concentrate on the Hybrid CMOS realization, for the time being.

(1) *AND-OR CMOS realization:* A CMOS complex gate realization of any function based

on its irredundant sum of products expression will be called an AND-OR CMOS realization.

Figure 4.1 shows the AND-OR CMOS realization of a small part of Marouf-Friedman's 2-out-of-5 checker [14]. It implements the function

$f = x_1x_3 + x_1x_4 + x_1x_5 + x_2x_3 + x_2x_4 + x_2x_5$

Note that another way of implementing this function would be to use uncomplemented inputs and follow the CMOS complex gate, formed with these inputs, by an inverter. Our subsequent results can be easily extended to this realization. For simplicity, however, we will stick to the realization given in Figure 4.1.

(2) *OR-AND CMOS realization:* A CMOS complex gate realization of any function based on its irredundant product of sums expression will be called an OR-AND CMOS realization.

For the same function f, Figure 4.2 shows the OR-AND CMOS realization corresponding to the product of sums expression $f = (x_1 + x_2)(x_3 + x_4 + x_5)$

(3) *Hybrid CMOS realization:* For any given function, if the nMOS network of its AND-OR CMOS realization is attached to the pMOS network of its OR-AND CMOS realization to form a CMOS complex gate, then such a realization will be referred to as a Hybrid CMOS realization.

It is easy to verify that this realization is also a viable way of implementing any function in the CMOS technology. In fact, it is possible to have valid Hybrid CMOS realizations by attaching the nMOS network of one realization with the pMOS network of another, if both realize the same function. We will consider some of these general

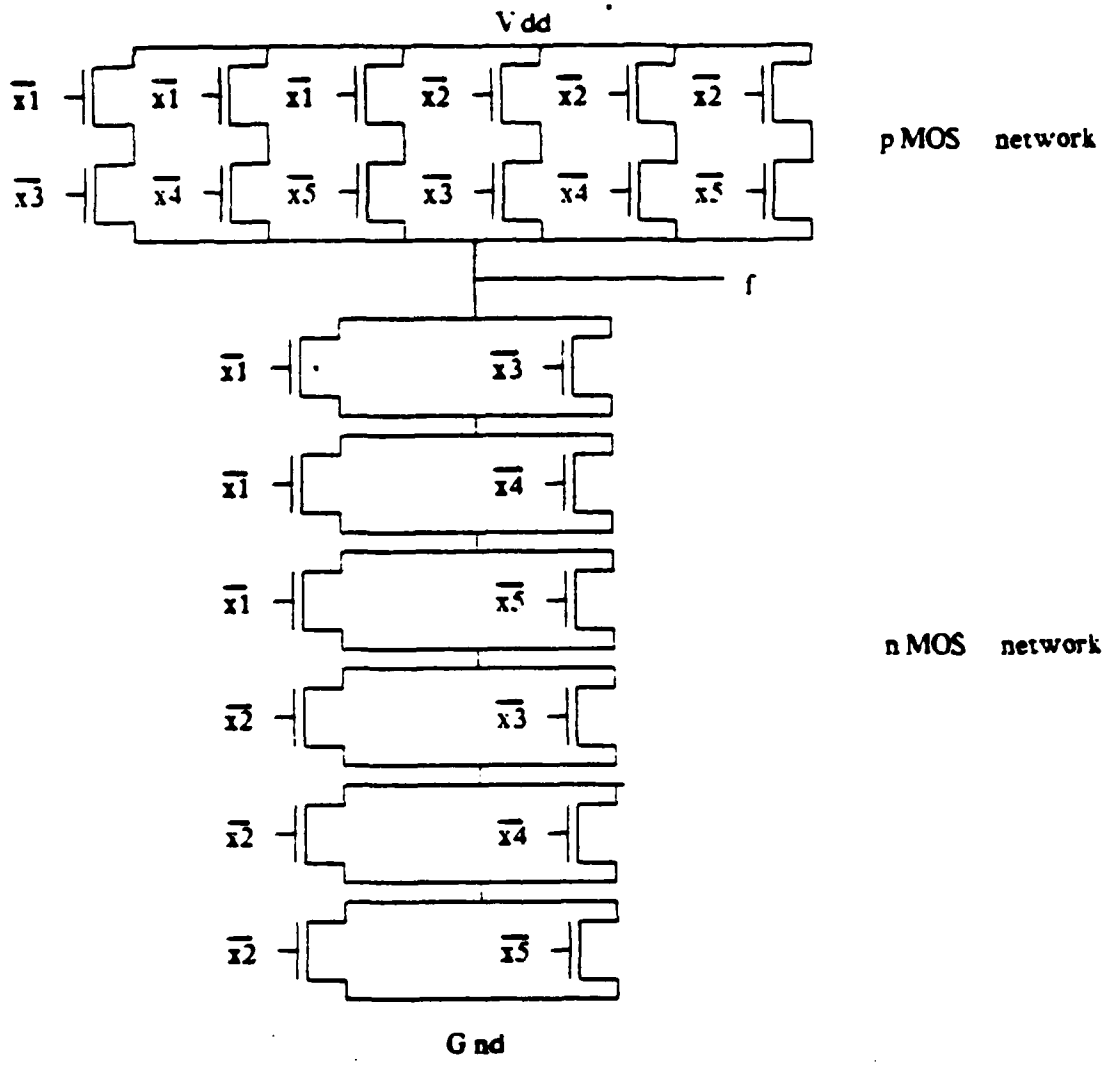


Figure 4.1 An AND-OR CMOS realization of f = x1.x3 + x1.x4 + x1.x5 + x2.x3 + x2.x4 + x2.x5

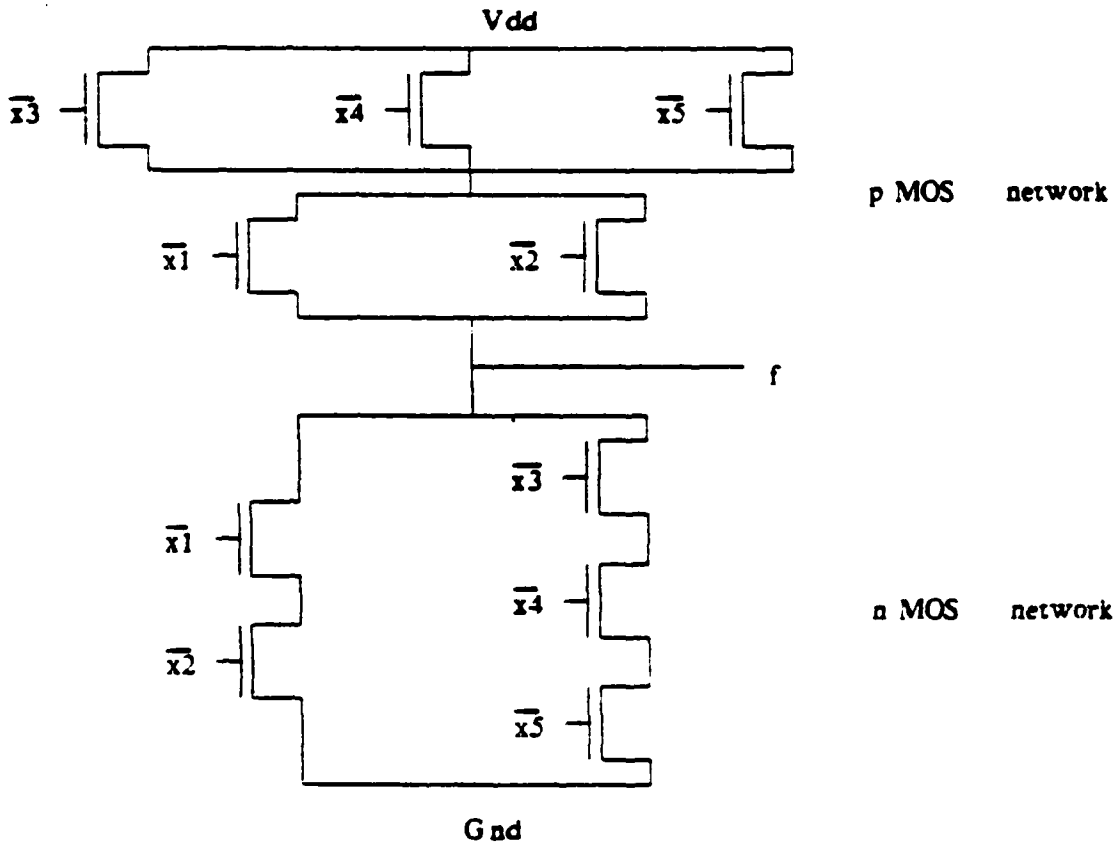


Figure 4.2 An OR-AND CMOS realization of f = (x1 + x2)(x3 + x4 + x5)

Hybrid CMOS realizations later.

Figure 4.3 shows the Hybrid CMOS realization for the function f mentioned above.

4.3. CMOS implementation of TSC circuits

It has been shown that AND-OR and OR-AND CMOS realizations of TSC circuits may not be self-testing with respect to stuck-open faults in the presence of arbitrary delays in the circuit [17]. We will use the Hybrid CMOS realization to implement the complex CMOS gates. Existing TSC circuits implemented in this fashion will be shown to be self-testing with respect to stuck-open faults, even in the presence of arbitrary delays in the circuit.

We next present a theorem which will show that the Hybrid CMOS realization is self-testing with respect to stuck-open faults in the presence of arbitrary delays.

Theorem 4.1: If the stuck-at faults in the two-level AND-OR and OR-AND gate-level realizations of each function, whose implementation requires a CMOS complex gate, are testable with the application of codewords at the primary inputs, then if all such functions are implemented with the Hybrid CMOS realization, the resultant CMOS circuit will remain self-testing with respect to stuck-open faults even in the presence of arbitrary delays through different gates and interconnections.

Proof: It is easy to see that in the nMOS (pMOS) network of the Hybrid CMOS realization there is a set of transistors connected in parallel, corresponding to each prime implicant (implicate) of the function. The number of such transistors in parallel is the same as the number of literals in the prime implicant (implicate). The inputs to these transistors are the complements of the literals in the prime implicant (implicate). These

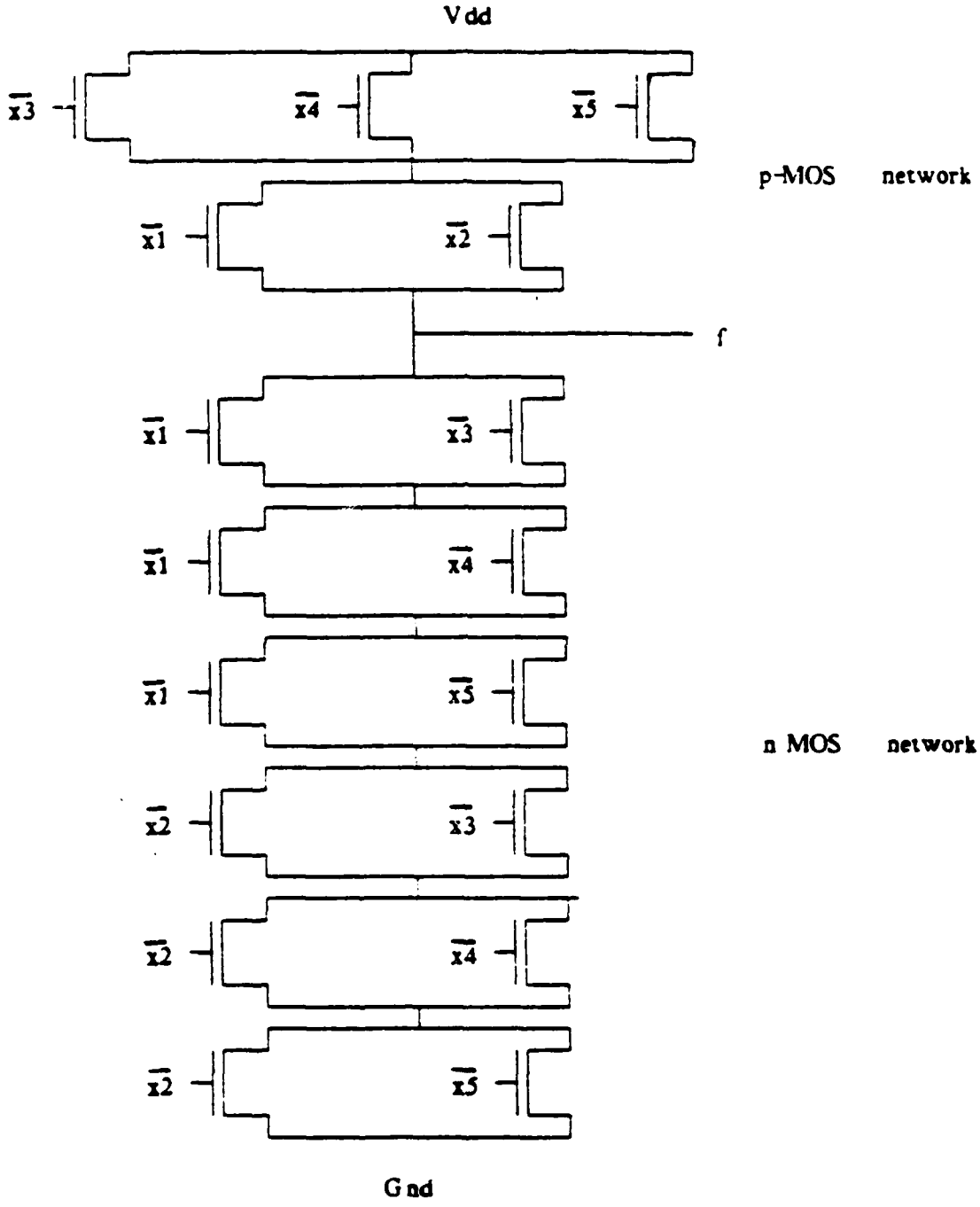


Figure 4.3 The Hybrid CMOS realization of f

sets of transistors are themselves connected in series.

Consider the nMOS network of the Hybrid CMOS realization of one of the functions, which requires a CMOS complex gate for its implementation. We know that the two-level AND-OR gate-level realization of this function is self-testing with respect to stuck-at faults. Now consider one of the AND gates in the first level of this gate-level AND-OR realization. The inputs of this AND gate correspond to a prime implicant (say P_i). Let the set of transistors connected in parallel, corresponding to P_i, in the nMOS network of the Hybrid CMOS realization be S_i. It is clear that the test (say X_i) which detects stuck-at-0 (s-a-0) faults on these inputs causes logic 0 to appear at the inputs of the transistors in S_i. This means that the output node has a logic 1, under normal operation, for this test. Now let us consider a test (say Y_i) which detects a stuck-at-1 (s-a-1) fault on one of the inputs of this AND gate. This test will cause a logic 1 to appear at the input of the corresponding transistor (say M_{ij}) in S_i and logic 0 at inputs of the other transistors in S_i. For this test the output is logic 0 in the fault-free case. Hence the two-pattern test (X_i-Y_i) detects the stuck-open fault in M_{ij}. This two-pattern test cannot be invalidated by arbitrary delays because only the transistor under test, namely M_{ij}, and no other transistor in S_i is activated. Trivially extending the above arguments, all the stuck-open faults in the nMOS network can be seen to be detectable in the presence of arbitrary delays.

We can treat the pMOS network in exactly the same fashion. Consider an OR gate in the first level of the gate-level OR-AND realization of the function. The inputs of this OR gate correspond to a prime implicate (say Q_k). Let the set of transistors connected in parallel, corresponding to Q_k, in the pMOS network of the Hybrid CMOS realization be

T_k. The test detecting a s-a-1 fault at the inputs of this OR gate causes a logic 1 to appear at the inputs of the transistors in T_k. Hence the output node is at logic 0 for this test in the fault-free case. The s-a-0 fault test for an input of this OR gate will cause a logic 0 to appear at the input of the corresponding transistor in T_k. For this test the output is logic 1 when no fault is present. So again these two tests together detect a stuck-open fault in that transistor without being invalidated by timing-skews. The extension of the above arguments to all transistors in the pMOS network is obvious.

It is easy to see that the test input of the two-pattern test, in the presence of a stuck-open fault, causes the same logic value to appear on the output node of the Hybrid CMOS realization, as would appear at the output of the AND-OR or OR-AND gate-level realization under a stuck-at fault. Hence it follows that the effect of applying the two-pattern test will be observable at the circuit outputs.

□

It is evident from Theorem 4.1 that the Hybrid CMOS realization is very useful for implementing TSC circuits. The important outcome of using this realization is that the CMOS implementation remains self-testing with respect to stuck-open faults even in the presence of arbitrary delays.

We will now present a Procedure outlining the steps in the design of a TSC CMOS circuit.

Procedure 4.1:

(1) Identify the different functions in the circuit which require a complex CMOS gate for their implementation.

(2) Check to see if application of codewords at primary inputs detects single stuck-at faults in the two level AND-OR and OR-AND gate-level realizations of these functions.

(3) If the condition in (2) is satisfied, implement these functions with Hybrid CMOS complex gates.

(4) Reduce the Hybrid CMOS gates by the method given in Section 4.3.1 ahead.

Now we will give an example to illustrate this procedure.

Example 4.1: Figure 4.4 shows the gate-level implementation of Marouf-Friedman's 2-out-of-5 checker [14]. The functions at f_1, f_3, y_1 and y_2 require a complex CMOS gate for their CMOS implementation. Hence these should be realized with Hybrid CMOS complex gates. The other functions require primitive CMOS gates for their implementation. These can be easily be shown to be degenerate forms of Hybrid CMOS gates. So we only need to concentrate on the CMOS complex gates.

Now the question arises as to how we can find out whether the AND-OR and OR-AND gate-level realizations of each function (which requires a complex gate for its implementation) is testable, with respect to stuck-at faults, with application of codewords at primary inputs. We will take help of the work done on universal or complete test sets [37-39], to answer this question. A universal test set for any function simply detects stuck-at faults in any realization of that function consisting of only AND and OR gates. We first find the universal test set for each such function. If this only consists of tests which are a result of application of codewords at the primary inputs, then the question is automatically answered. This is found to be true in most cases. But in some cases this is not true. For example, let us consider the function f implemented in

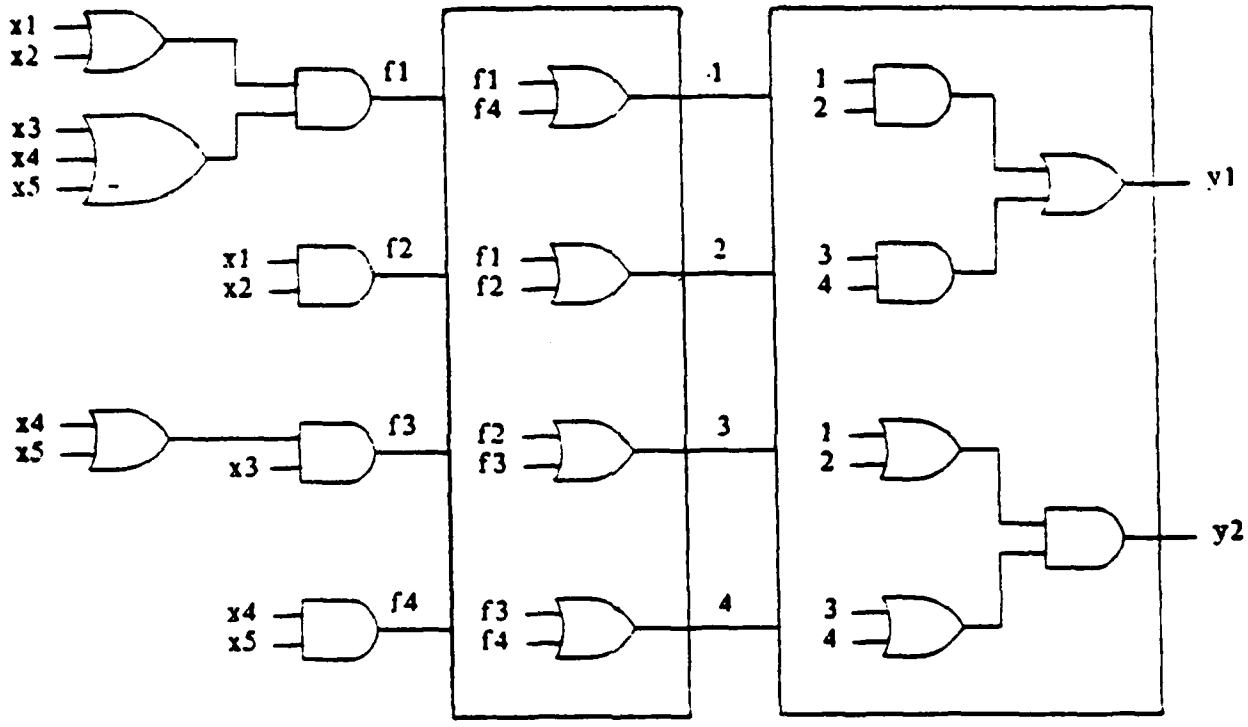


Figure 4.4 Marouf-Friedman's 2-out-of-5 code checker

Figures 4.1, 4.2 and 4.3. This function is the same as function f_1 of Marouf-Friedman's 2-out-of-5 checker shown in Figure 4.4. Let S_0 be a test set that detects s-a-0 faults in any implementation of f with AND and OR gates, and S_1 be an analogous test set for detecting s-a-1 faults. Then for f we have

$S_0 = (10100, 10010, 10001, 01100, 01010, 01001)$

$S_1 = (00111, 11000)$

The union of S_0 and S_1 is the universal test set. We find that the test "00111" in S_1 does not belong to a 2-out-of-5 code. But if we replace this test with the 2-out-of-5 codewords that it covers, namely (00110,00101,00011), then one can easily see that the union of S_0 and the modified S_1 is still a universal test set, although no longer minimal. So the function f_1 satisfies the condition in Theorem 4.1 and hence its Hybrid CMOS realization, given in Figure 4.3, is self-testing with respect to stuck-open faults, even in the presence of arbitrary delays. One can easily verify that the stuck-open faults are also detectable at the outputs y_1 and y_2 when this Hybrid CMOS realization is used in the CMOS implementation of the circuit in Figure 4.4.

Now we will move on to investigate the self-testing property of the TSC circuits with respect to stuck-on faults. The problem of detecting stuck-on transistors was addressed in [32]. These faults cause a transistor to be always conducting. The test which can detect a stuck-on transistor does not activate any conduction path in the network to which that transistor belongs in the fault-free case, and activates a conduction path in the presence of a faulty stuck-on transistor. Therefore both the nMOS and pMOS networks of the CMOS complex gate conduct under fault. The result

is that the output node assumes a value between logic 0 and logic 1 and may be interpreted either way by the subsequent level of CMOS gates. This means that this fault may or may not be detected by just monitoring logic levels. One solution to this is suggested in [36], where after applying the appropriate test as above, the steady state current through the circuit is measured. This would be large in the faulty circuits compared to the fault-free case and thus the fault can be detected.

Reducing a Hybrid CMOS realization

If the Hybrid CMOS realization seems unwieldy, there is, fortunately, a way of reducing it without affecting its self-testing property. To explain the reduction procedure we will consider the nMOS network only. Exactly similar arguments are applicable to the pMOS network. We have seen that both nMOS and pMOS networks of a Hybrid CMOS realization consist of a series connection of a set of transistors connected in parallel. If any transistor from a set of parallel transistors has a common input with a transistor(s) from any other set, then these transistors can be reduced to one, following the law $(x+y)(x+z)=(x+yz)$, from Switching Algebra, where x, y and z can be variables or sub-expressions.

For example, from Figure 4.3 we get

$\bar{f}=(\bar{x}_1+\bar{x}_3)(\bar{x}_1+\bar{x}_4)(\bar{x}_1+\bar{x}_5)(\bar{x}_2+\bar{x}_3)(\bar{x}_2+\bar{x}_4)(\bar{x}_2+\bar{x}_5)$, which can, at first be reduced to $\bar{f}=(\bar{x}_1+\bar{x}_3\bar{x}_4\bar{x}_5)(\bar{x}_2+\bar{x}_3\bar{x}_4\bar{x}_5)$. Now extending this concept to a series of transistors, we have $\bar{f}=(\bar{x}_1\bar{x}_2+\bar{x}_3\bar{x}_4\bar{x}_5)$. We will now give a theorem to show that the stuck-open faults in the Reduced Hybrid CMOS realization are also detectable in the presence of arbitrary

delays.

Theorem 4.2: The test set which detects the stuck-open faults in a Hybrid CMOS realization in the presence of arbitrary delays is sufficient to detect stuck-open faults in the Reduced Hybrid CMOS realization.

Proof: We will consider only the nMOS network of the Hybrid CMOS realization here. Similar arguments are applicable to the pMOS network.

The Hybrid CMOS realization is reduced by repeatedly using the law $(x+y)(x+z)=(x+yz)$, as mentioned above. Here x, y and z can be variables or sub-expressions. Let us first assume that x is a variable. Now let us consider the set of transistors connected in parallel, which correspond to $(x+y)$. Referring to the Proof of Theorem 4.1, we see that the transistor, to which x is fed, is tested by first letting $x=y=0$ and then making $x=1$ while $y=0$. This enables the detection of a stuck-open fault in this transistor without the test being invalidated due to arbitrary delays. This two-pattern test is a valid test for the Reduced Hybrid CMOS realization also. We can say this by observing that, since $y=0$ for the initialization and test inputs, the series connection $y.z$ will remain disabled for both inputs. Thus the transistor fed by x will be tested for a stuck-open fault, even in the presence of arbitrary delays, as before. Since the transistor being fed by the variable x in the set of transistors corresponding to the sub-expression $(x+z)$ is no longer there, its two-pattern test is not required. Hence the test set for testing the Reduced Hybrid CMOS realization is a subset of the test set required for testing the Hybrid CMOS realization. It can easily be seen that the same arguments would be applicable if x were a sub-expression instead of a variable.

□

It is interesting to note that we obtain the OR-AND CMOS realization, given in Figure 4.2, after reducing the Hybrid CMOS realization in Figure 4.3. Frequently, a Hybrid CMOS realization is reducible to an OR-AND or AND-OR CMOS realization for TSC circuits. In such cases only the corresponding OR-AND or AND-OR gate-level realization need to be checked to see if it is testable with respect to stuck-at faults, but not both the realizations. Note that the condition in Theorem 4.1 is required to be satisfied for Hybrid CMOS realizations, not necessarily for Reduced Hybrid CMOS realizations. This condition can be relaxed for Reduced Hybrid CMOS realizations which are same as the AND-OR or OR-AND CMOS realization, as explained above.

CHAPTER 5

EFFICIENT MOS IMPLEMENTATION OF TSC CHECKERS

5.1. Introduction

In order to pack as many circuits on a single chip as possible, it is essential to reduce the area requirement of each circuit. One way to do this is to implement the circuit with as few transistors as possible. In this chapter we will show how this can be done for TSC checkers.

It has been pointed out in [16] that a MOS implementation cannot be TSC with respect to unidirectional stuck-at faults (multiple lines stuck-at 1 or stuck-at 0, but not both). This is owing to a Theorem in [15], which says that in order for a circuit to be TSC with respect to unidirectional stuck-at faults its realization should be inverter-free. This is not possible in any MOS technology since every MOS gate is inverting. However, MOS implementations of TSC circuits can be made TSC with respect to single stuck-at faults. So for the purpose of this chapter the fault-set will consist of all *single* stuck-at faults. Techniques developed in Chapter 3 can be used to make these MOS implementations TSC with respect to realistic physical failures.

A direct implementation of existing TSC designs in MOS technology requires a high transistor count and, therefore, a large silicon area. We will present some techniques in this chapter to reduce the transistor count, with the added advantages of speed-up of the circuit and reduction in the number of tests required. Note that reducing transistor count reduces area even further because of fewer interconnections. In [16] the technological cost (in case of nMOS technology) of different coding and checking circuits has been evaluated. We will show that by using our techniques one can obtain marked

improvements over the results given in [16]. Although our techniques can be used for any MOS circuit, wherever applicable, we will show their usefulness by applying them to TSC m-out-of-n and Berger code checkers. Both these codes detect unidirectional errors. It should be kept in mind that even single stuck-at faults can produce unidirectional errors. The m-out-of-n code consists of a set of n-bit codewords which have exactly m 1's and n-m 0's. A Berger code consists of codewords with separable information bit and checkbit parts. The checkbits represent the number of 0's in the information part.

To take full advantage of the integration capability of MOS technology it is essential to have as many control transistors in every complex MOS gate as possible. However, arbitrary positioning of inverting functions and fanouts restricts the integration that can be achieved. In the next section we present some techniques which ease this restriction.

5.2. Reducing the cost of TSC checkers

5.2.1. Transfer of fanouts

The presence of functions, which are shared by more than one MOS gate (i.e., when there is fanout), necessitates the use of a complex gate for their MOS realizations. In Example 5.1 below we will see how transfer of fanouts can help achieve greater integration. The techniques developed here are first applied to existing gate-level designs of TSC checkers, thereby obtaining modified gate-level designs, which can then be implemented in MOS technology, with a resultant decrease in their technological cost. Hence the examples we will present will be of gate-level circuits, rather than their MOS implementations. We assume that an AND (OR) gate in a gate-level realization is implemented as a series (parallel) connection of transistors in a complex MOS gate. This

complex MOS gate is followed by an inverter to get the non-inverting function.

Example 5.1: The fanout in the circuit in Figure 5.1(a) has been transferred to the primary inputs in the circuit in Figure 5.1(b). Figures 5.2(a) and 5.2(b) show the respective nMOS implementations.

While the circuit in Figure 5.2(a) requires 6 pull-up transistors and 9 control transistors for its MOS realization, the circuit in Figure 5.2(b) requires only 4 pull-up transistors and 8 control transistors. Thus transfer of this fanout helps reduce the transistor count of the circuit.

The transfer of fanouts can sometimes make the circuit redundant. Fortunately this is not true in general, and impressive reductions in transistor count can be achieved by judicious transfer of fanout. It can also result in the loss of the self-testing property of a TSC circuit. So some conditions need to be developed to ensure that the self-testing property is maintained even after the transfer of fanouts. We will develop these conditions for m-out-of-n code checkers. The same concept can be carried over to other circuits as well. But before we do so, let us familiarize ourselves with some definitions and notations.

An r-input network is said to have 2^r vertices of the r-cube as possible input combinations. A particular vertex of the r-cube is written as

$$A = (a_1, a_2, \cdots, a_r) \text{ where } a_i \text{ belongs to } \{0,1\}.$$

A vertex with exactly k 1's is called a k-vertex.

Definition 5.1: The partial ordering on the vertices is defined as

$$A \leq B \text{ iff } a_i \leq b_i \text{ for all i}$$

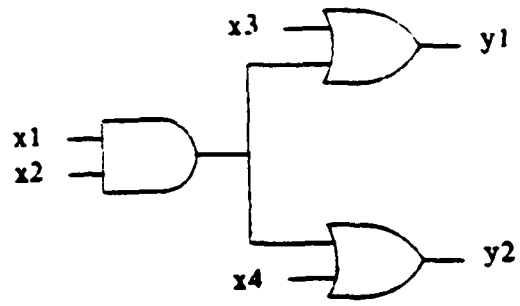


Figure 5.1(a) A gate-level circuit with fanout

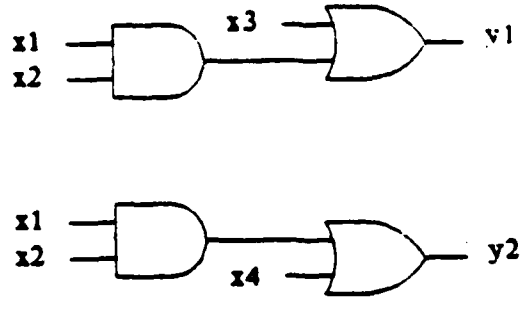


Figure 5.1(b) The circuit with fanout transferred to primary inputs

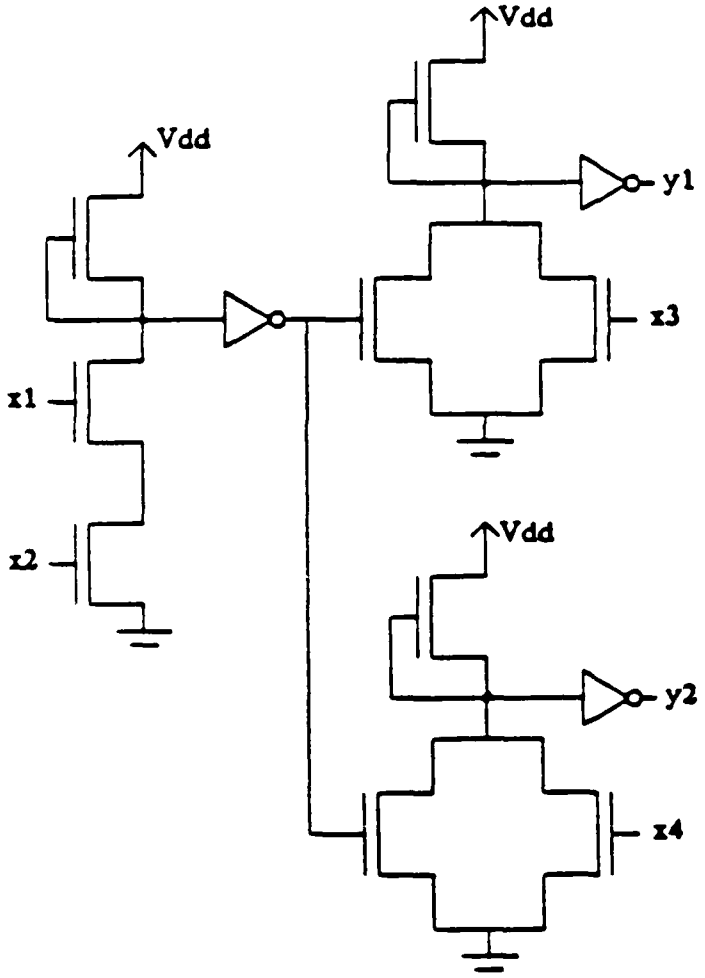


Figure 5.2(a) nMOS implementation of the circuit in Figure 5.1(a)

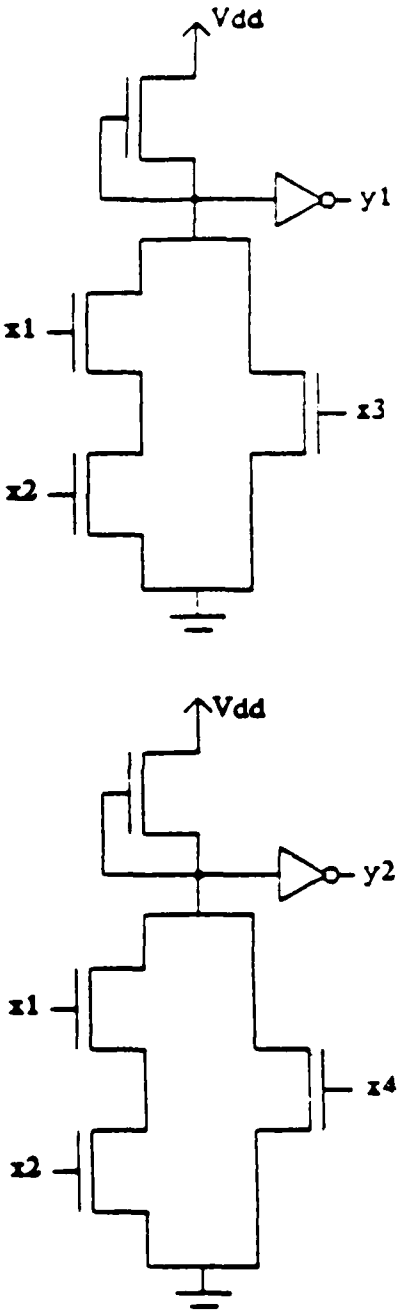


Figure 5.2(b) nMOS implementation of the circuit in Figure 5.1(b)

For example,

$$(1,1,0,0) \leq (1,1,0,1)$$

$$(1,0,0,1) \not\leq (1,1,0,0)$$

We will say that A covers B (or B is covered by A) if B $\leq$ A. We will say that vertex A is adjacent to vertex B if A $\leq$ B and there does not exist a vertex C such that A $\leq$ C $\leq$ B.

We have already defined a universal test set in Chapter 4. The procedure for finding the universal test set is given in [37]. We will describe it briefly here.

Procedure 5.1:

(1) Construct a truth table having one column for each literal in the functional expression.

(2) Let A_i denote an input combination for which f is 1 and B_j an input for which f is 0. For all A_k, B_l, if

$$A_k \geq A_i$$

$$B_j \geq B_l$$

remove A_k and B_l from the table.

(3) Repeat Step 2 until no more covering A's or covered B's remain. The set of input combinations left in the table constitutes the universal test set for that function.

Theorem 5.1 below gives sufficient conditions for a set of m-out-of-n code words to

be a universal test set.

Theorem 5.1: In a mapping of an m-out-of-n code onto a 1-out-of-2 code (realized by functions y_1 and y_2) if

(1) each (m+1)-vertex covers some m-vertex mapping to (1,0) and some m-vertex mapping to (0,1) and

(2) each (m-1)-vertex is covered by some m-vertex mapping to (1,0) and some m-vertex mapping to (0,1),

then the set of codewords constituting the m-out-of-n code is the universal test set for both y_1 and y_2.

Proof: From the monotone property of the circuit constructed with only AND/OR gates and the code-disjoint property of the checker, all (m+1) and higher vertices will map to (1,1), and all (m-1) and lower vertices will map to (0,0). Since Condition (1) ensures that every (m+1)-vertex covers some m-vertex mapping to (1,0), the (m+1)-vertices cannot belong to the universal test for testing any irredundant AND/OR implementation of the function y_1 (see Procedure 5.1); and since every (m+1)-vertex covers some m-vertex mapping to (0,1), the same is true for the function y_2. Similarly, since every (m-1)-vertex is covered by some m-vertex mapping to (1,0), the (m-1)-vertices cannot belong to the universal test set for the function y_2; and since every (m-1)-vertex is also covered by some m-vertex mapping to (0,1) the same is true for the function y_1.

We have proved above that, given Conditions (1) and (2), any (m+1)-vertex or (m-1)-vertex cannot belong to the universal test set of either function y_1 or y_2. Now we will consider the (m-2) and lower vertices. All such vertices map to (0,0). Due to the

transitivity of the covering relation, for every (m-2) or lower vertex X_i there exists an (m-1)-vertex X_j such that $X_i \leq X_j$. But from Condition (2), we have, $X_j \leq X_k$ and $X_j \leq X_l$, where X_k is some m-vertex mapping to (1,0) and X_l is some m-vertex mapping to (0,1). This implies that $X_i \leq X_k$ and $X_i \leq X_l$. So, from Procedure 5.1, any (m-2) or lower vertex also cannot belong to the universal test set of either y_1 or y_2. A similar argument can be used to show that any (m+2) or higher vertex cannot belong to the universal test set of either y_1 or y_2.

Since every m-vertex has the property that it does not cover any other m-vertex, only these m-vertices constitute the universal test set for both y_1 and y_2.

□

An alternative proof of Theorem 5.1 can be obtained by observing that the m-vertices are either minimal true-vertices or maximal false-vertices for y_1 and y_2, and, hence, constitute the complete test set [39].

This theorem implies that if the input codewords constitute the universal test set for the two output functions of the TSC checker, then any fanout in the checker can be transferred without fear of losing the self-testing property.

Theorem 5.1 is valid for Smith's m-out-of-2m checkers [15] since they are based precisely on the two conditions given in this theorem.

We will now focus our attention on Reddy's m-out-of-2m checkers [40]. Reddy gave a multi-level cellular realization for the checker function y_1 and y_2 for an m-out-of-2m checker as follows:

$$y_1 = \sum_{i=0}^{m} \mathrm{T}(m_a \geq i).\mathrm{T}(m_b \geq m-i), \qquad \text{i odd}$$

$$y_2 = \sum_{i=0}^{m} \mathrm{T}(m_a \geq i).\mathrm{T}(m_b \geq m-i), \qquad \text{i even}$$

Here the input bits are divided into two groups A and B, each consisting of m bits. The number of 1's occurring in each group is referred to as m_a and m_b. For code inputs $m_a + m_b = m$.

We will say that a k-vertex belongs to a class represented by a 2-tuple (k_a, k_b) if it has k_a 1's in group A and k_b 1's in group B, and $k_a + k_b = k$.

Theorem 5.2: The set of m-out-of-2m codewords constitutes a universal test set for y_1 and y_2 functions of Reddy's checkers given in [40].

Proof: If we prove that Reddy's y_1 and y_2 functions satisfy Conditions (1) and (2) in Theorem 5.1 then this theorem will follow.

We will first consider the (m+1)-vertices. In the first column of Table 5.1 we give all different classes of 2-tuples to which any (m+1)-vertex can possibly belong. The entries in the second column give the classes to which the m-vertices covered by that (m+1)-vertex belong.

Table 5.1 Classes of (m+1)-vertices and m-vertices they cover

Possible classes of (m+1)-vertices	Covered m-vertices belong to
(m,1)	(m,0), (m-1,1)
(m-1,2)	(m-1,1), (m-2,2)
(m-2,3)	(m-2,2), (m-3,3)
-	-
-	-
-	-
(2,m-1)	(2,m-2), (1,m-1)
(1,m)	(1,m-1), (0,m)

It can easily be verified that the m-vertices belonging to the two different 2-tuples appearing in the second column of Table 5.1, for any given (m+1)-vertex, map to the two different outputs (0,1) and (1,0). Hence Condition (1) of Theorem 5.1 is satisfied.

Similarly, Table 5.2 can be formed for (m-1)-vertices.

Table 5.2 Classes of (m-1)-vertices and the m-vertices that cover them

Possible classes of (m-1)-vertices	covering m-vertices belong to
(m-1,0)	(m,0), (m-1,1)
(m-2,1)	(m-1,1), (m-2,2)
(m-3,2)	(m-2,2), (m-3,3)
-	-
-	-
-	-
(1,m-2)	(2,m-2), (1,m-1)
(0,m-1)	(1,m-1), (0,m)

Following the same arguments as above, Condition (2) of Theorem 5.1 is seen to be satisfied.

Hence Theorem 5.2 follows from Theorem 5.1.

□

The implication of Theorem 5.2 is that some or all the fanouts in Reddy's m-out-of-2m code checkers can be judiciously transferred to some intermediate levels in the circuit. This does not make the circuit redundant, and allows us a lot of flexibility in the MOS implementation of these checkers.

It can be easily seen that the code-disjoint property is not affected by fanout transfer because the function being implemented is still the same.

Example 5.2: If we transfer all the fanouts in Reddy's 5-out-of-10 code checker [40] to the primary inputs, the nMOS implementation requires 4 pull-up transistors and 116 control transistors. But another nMOS implementation, which leaves four of the fanouts untransferred, requires only 12 pull-up transistors and 80 control transistors. If we had implemented this checker without modification, it would require 40 pull-up transistors and 88 control transistors. So the savings in the transistor count is evident, as is the flexibility that this technique allows.

Another advantage of this technique is that it reduces the MOS gate levels from inputs to outputs and, hence, increases the speed of the circuit. On the other hand the disadvantage is that more tests are required to make the circuit self-testing. For example, for Reddy's checkers the number of tests required goes up from 2m to 2^m, if all fanouts are transferred to primary inputs. If all the fanouts were not transferred then the number of tests lies between these two limits. The 2^m tests, in case of full fanout transfer, are the same as those required for Anderson's m-out-of-2m code checkers. Anderson's m-out-of-2m code checkers [11] do not have fanouts, hence this technique is

not applicable to them. Although this technique increases the number of required tests, when it is combined with the techniques given ahead, there is usually a considerable reduction in the size of the test set.

Now we go on to our second technique.

5.2.2. Removal of inverters

A function, which is shared by more than one MOS gate, is realized by a complex MOS gate followed by an inverter. We will show that these inverters can be eliminated in most cases, thereby reducing the transistor count and increasing the speed of the MOS checker. In [16] it is mentioned that such inverters can be eliminated if the function located after a fanout node can be replaced by its dual function without modifying the function realized by the circuit. We will not restrict ourselves to such functions.

Before going ahead let us present some definitions. Let G(f) be an AND/OR realization of a function f and let $G_{dm}(f)$ be the corresponding realization of f after using De Morgan's theorem. For example, if G(f) realizes $f = x_1x_2 + x_3x_4$, then $G_{dm}(f)$ realizes $f = (\overline{\bar{x}_1 + \bar{x}_2})(\overline{\bar{x}_3 + \bar{x}_4})$. G(f) and $G_{dm}(f)$ are shown in Figure 5.3(a) and Figure 5.3(b) respectively.

A standard way of generating $G_{dm}(f)$ from G(f) is given by Procedure 5.2.

Procedure 5.2

(1) Replace every AND (NAND) gate in G(f) by an OR (NOR) gate in $G_{dm}(f)$ and every OR (NOR) gate in G(f) by an AND (NAND) gate in $G_{dm}(f)$

(2) Replace all primary inputs by their complements

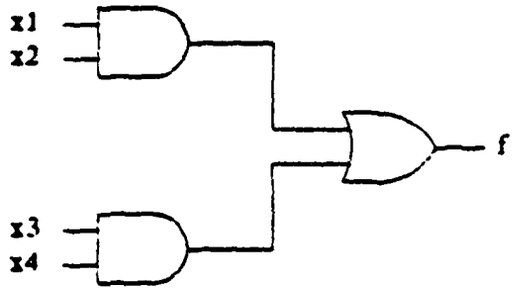


Figure 5.3(a) A gate-level circuit realizing f = x1.x2 + x3.x4

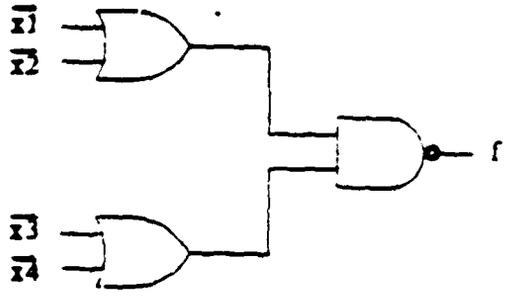


Figure 5.3(b) The circuit after appying De Morgan's theorem

(3) Add an inverter at the circuit output.

We have omitted an intermediate step in Procedure 5.2 that requires two inverters to be put on every line in G(f) and then a NAND gate to be replaced with an OR gate with inputs complemented, and a NOR gate with an AND gate with inputs complemented, or vice-versa. This intermediate step is shown in Figure 5.4. After the above conversions we get Figure 5.3(b).

Theorem 5.3: For any function f, G(f) and $G_{dm}(f)$ have the same stuck-at fault test set.

Proof: Let the realization obtained by placing two inverters on every line of G(f) be $G_i(f)$. The stuck-at fault test set for $G_i(f)$ is the same as that of G(f), because the path sensitization of the stuck-at faults by their corresponding tests remains unaffected. Furthermore, it is easy to see that corresponding lines of the two gates in Figure 5.5(a) and in Figure 5.5(b) require the same test set for detecting the presence of the stuck-at faults.

The two gates in Figure 5.5(a) and in Figure 5.5(b) can therefore be interchanged in the circuit $G_i(f)$ without changing its test set. From Procedure 5.2 and its intermediate step given above, it can easily be seen that $G_{dm}(f)$ can be obtained from $G_i(f)$ by these interchanges of the two gates. Hence, since $G_i(f)$ has the same test set as G(f), so will $G_{dm}(f)$ and G(f).

□

This theorem implies that the circuits in Figures 5.3(a), 5.3(b) and 5.4 have the same stuck-at fault set, which can be easily verified. Hence by converting G(f) to $G_{dm}(f)$

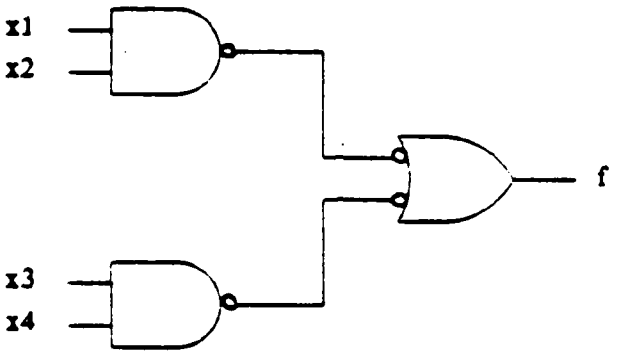


Figure 5.4 Intermediate step in the conversion of Figure 5.3(a) to 5.3(b)

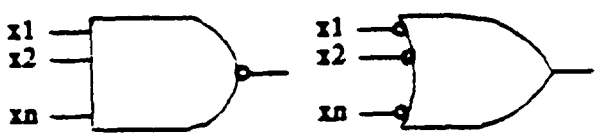


Figure 5.5(a) A NAND gate and its equivalent

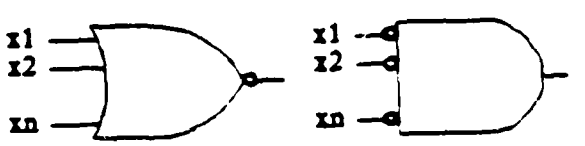


Figure 5.5(b) A NOR gate and its equivalent

we do not affect the self-testing property of the circuit.

Example 5.3: Consider Marouf-Friedman's design of a 2-out-of-5 checker [14] given in Figure 4.4. This can be converted to Figure 5.6 without affecting its TSC property with respect to single stuck-at faults, with the resultant saving of 10 inverters in the MOS implementation. It is clear that the two inverters before the circuit outputs can also be done away with since the pair $(\bar{y}_1,\bar{y}_2)$ also forms a 1-out-of-2 code.

This technique also has the added advantage that it speeds up the MOS implementation by reducing the MOS gate levels from circuit inputs to outputs. In Example 5.3, the number of MOS gate levels in the MOS realization of Figure 4.4 is six, while for the MOS realization of Figure 5.6 it is three. Hence the speed is roughly doubled by using this technique for this example. This technique is applicable to designs in [14-15,41-43].

Now we continue on to our third technique.

5.2.3. Increasing logic gate levels before MOS implementation

It is possible to reduce the transistor count of an MOS implementation by modifying the expression of the given function. This was discussed in Chapter 3. It was shown there that it is desirable to first reduce the number of literals in the expression of a function before implementing it in an MOS technology. This basically corresponds to increasing the number of logic gate levels in a gate-level design before the MOS implementation. It was shown that this technique does not degrade the speed of the circuit. The added advantage is that the number of tests required to test the circuit is considerably reduced.

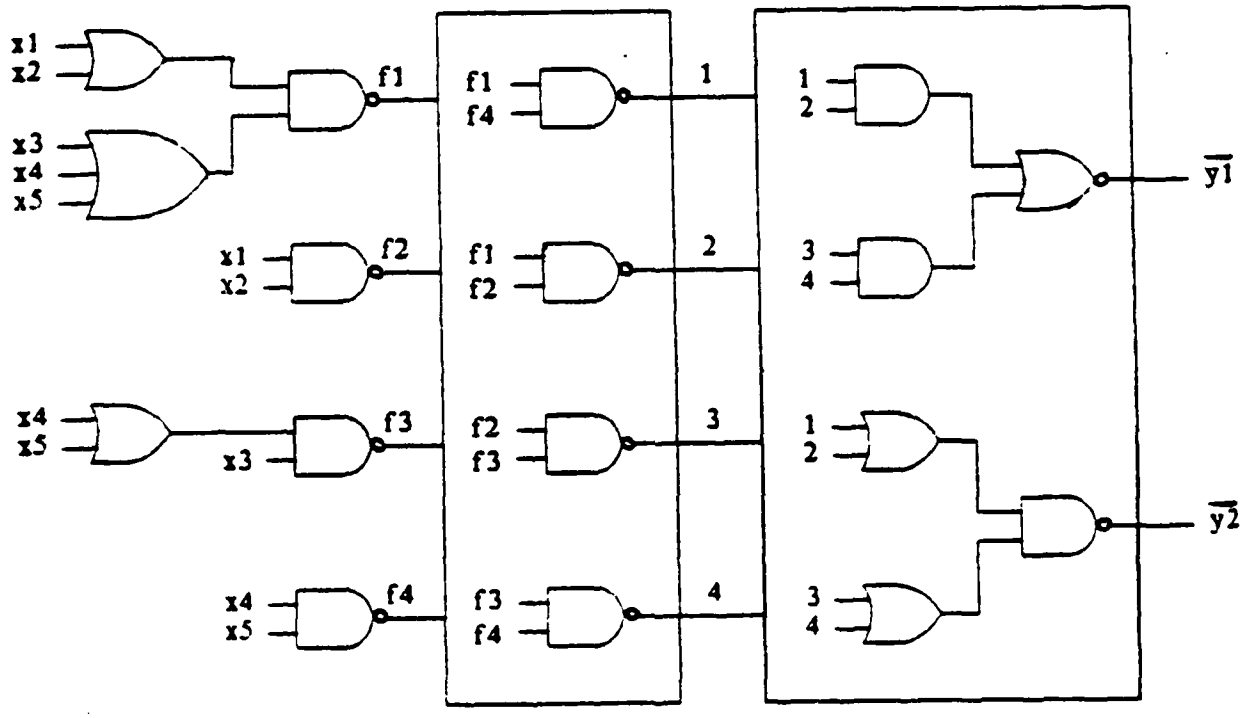


Figure 5.6 Marouf-Friedman's 2-out-of-5 code checker after using our technique

We will show the usefulness of this technique by applying it to MOS implementations of majority functions. Majority functions (defined below) are extensively used in the design of m-out-of-n code checkers.

Maximum level majority function realizations

We will first define a majority function. Let A be the set of input bits and let n_a and k_a represent the number of bits and the number of 1's occurring in the set respectively. The majority function $T(k_a \geq i)$, defined on set A, has a value 1 iff the condition inside the parentheses is true. For example, if $A = (a_1, a_2, a_3, a_4)$, then one realization of the majority function $T(k_a \geq 2)$ is as follows:

$$T(k_a \geq 2) = a_1a_2 + a_1a_3 + a_1a_4 + a_2a_3 + a_2a_4 + a_3a_4. \qquad (1)$$

This is a two level realization of $T(k_a \geq 2)$. Usually the m-out-of-n code checker designs employ majority function realizations with minimum number of levels due to speed considerations. But as was mentioned above, this thinking is no longer valid for MOS circuits, and one should use minimum literal or maximum level realizations to reduce the transistor count. An efficient way of obtaining maximum level majority function realization is given in [11,43-44]. For completeness we will briefly mention that method below.

Let the number of bits and the number of 1's in a set A, consisting of input bits, be n_a and k_a respectively. Divide the bits of the set A into two subsets A_1 and A_2, each subset containing at least one bit. Let the number of bits in A_1 and A_2 be n_{a_1} and n_{a_2} and the number of 1's occurring in A_1 and A_2 be k_{a_1} and k_{a_2} respectively. Obviously, $n_a = n_{a_1} + n_{a_2}$ and $k_a = k_{a_1} + k_{a_2}$.

Now $T(k_a \geq i)$ can be realized as

$$T(k_a \geq i) = \sum_{j=r}^{s} T(k_{a_1} \geq j) T(k_{a_2} \geq i - j) \qquad (2)$$

where $r = \max(i - n_{a_2}, 0)$ and $s = \min(n_{a_1}, i)$.

Let us consider the example with $A = (a_1, a_2, a_3, a_4)$ again. Let $A_1 = (a_1, a_2)$ and $A_2 = (a_3, a_4)$. Using the above equation we see that

$$T(k_a \geq 2) = a_1 a_2 + (a_1 + a_2)(a_3 + a_4) + a_3 a_4 \qquad (3)$$

Since the number of transistors in the MOS implementation has one-to-one correspondence with the number of literals in the function's expression, we see that the transistor count, from (1) to (3), has been reduced by four.

To achieve more levels the above procedure can be repeated until no subset contains more than two bits. So (3) also happens to be the minimum-literal, maximum-level realization of $T(k_a \geq 2)$. The dual of the above methods helps realize two-level OR-AND majority circuits or their corresponding multi-level circuits.

One question still remains, however; that is, whether the stuck-at fault test set of a two-level realization of any function (not just a majority function) can still be valid for a multi-level realization. This problem was briefly addressed for the general case in Chapter 3. We will give a Theorem below to show that this is indeed the case.

Theorem 5.4: The stuck-at fault test set S of a two level realization of any function suffices as a test set for its multi-level realization.

Proof: We will prove the theorem for the AND-OR realization. Dual arguments can be applied to the OR-AND realization.

Let $E_1(f)$ denote the functional expression of the two-level AND-OR realization and let $E_2(f)$ denote the functional expression of the multi-level realization of a function f. $E_1(f)$ is basically a sum of products expression of the function f.

If we expand $E_2(f)$ by removing the parentheses, we will arrive at $E_1(f)$. Conversely, we can factor out common literals or sub-expressions from the terms of $E_1(f)$ and the other intermediate expressions to arrive at $E_2(f)$. The process of conversion repeatedly applied, going from $E_1(f)$ to $E_2(f)$, is illustrated by Figures 5.7(a) and 5.7(b).

z_i in Figure 5.7(a) or 5.7(b) can be a literal or a sub-expression. It can easily be seen that the stuck-at fault test set for Figure 5.7(a) will suffice as a stuck-at fault test set for Figure 5.7(b) as well. Since $E_2(f)$ is obtained by repeatedly applying the above conversion at different levels, the theorem follows as a straightforward generalization.

□

The set S is sufficient, but not necessary, for making the multi-level realization self-testing. Usually a subset of S is required for this purpose. Hence this technique reduces the number of tests as well as the transistor count for MOS implementations. We can apply this technique to designs in [11,14,41-44].

5.3. Efficient MOS implementation of Berger code checkers

The Berger code, as mentioned in Section 5.1, is a separable code used to detect unidirectional errors. Design of gate level Berger code checkers is given in [13]. This design basically employs a combinational circuit which generates the complement of the checkbits. This circuit is realized with full adders and half-adders. The outputs of this combinational circuit, together with the checkbit lines from the primary inputs, are fed to a two-rail code checker.

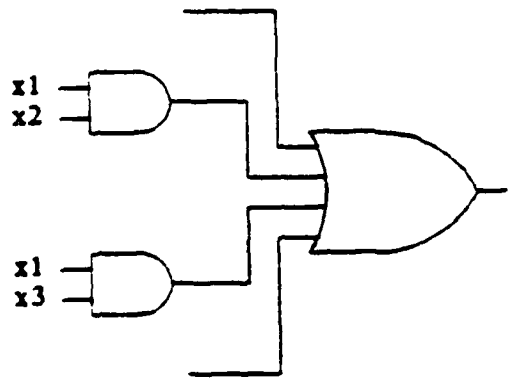


Figure 5.7(a) Two level realization of a function f

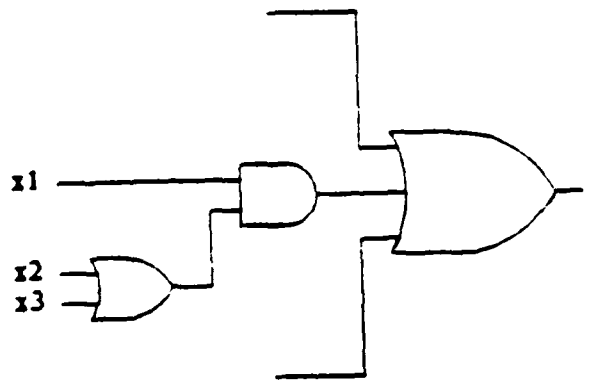


Figure 5.7(b) Multi-level realization of f

In [16] one way of implementing full adders and half-adders in MOS technology is given. The transistor count n_T (for nMOS technology) for the Berger code checker, on the basis of this implementation, is given as

$$n_T = 23(n_I - n_k) + 10(n_k - 1) = 23n_I - 13n_k - 10 \qquad (4)$$

where

n_I = number of information bits, and

n_k = number of checkbits

The number $23(n_I - n_k)$ refers to the transistor count of the checkbit complement generator and $10(n_k - 1)$ to the transistor count of the two-rail code checker.

We present below a more efficient way of implementing Berger code checkers. Figure 5.8 shows a gate level realization of the two-bit adder (with outputs complemented), whose MOS implementation requires only 14 transistors (12 control and 2 load transistors). Figure 5.9 shows a gate level realization of a two-bit adder to obtain uncomplemented outputs when complemented inputs are available. This also requires 14 transistors for its MOS implementation. The MOS implementation in [16] requires 23 transistors for a two-bit adder for obtaining uncomplemented outputs from uncomplemented inputs.

Let us now consider a Berger code with $n_I = 7$. We know that for the Berger code for this case, $n_k = 3$. Figure 5.10 shows the combinational circuit for generating the complement of the three checkbits, when Technique 5.2.2 is used for removal of inverters and efficient realizations given in Figures 5.8 and 5.9 are used for the two-bit adders. If the checkbit complement generator is combined with a two-rail code checker to obtain a Berger code checker, we get the following equation for the transistor count of the Berger

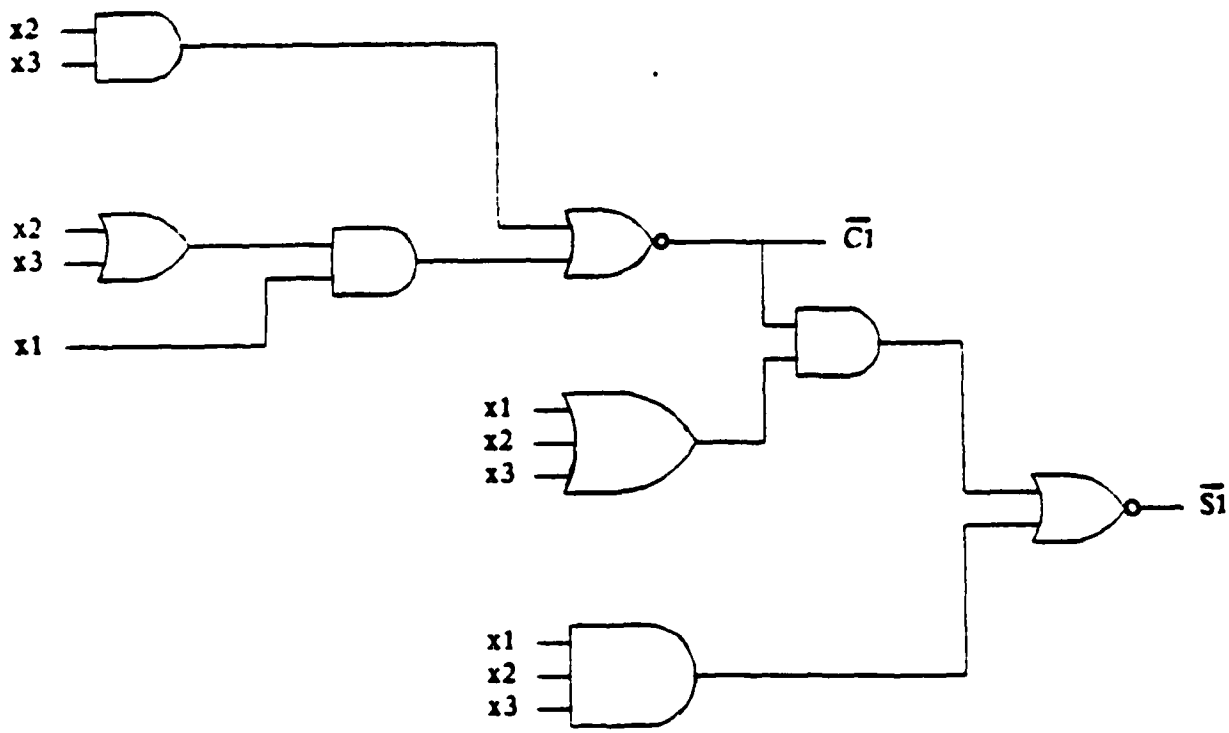


Figure 5.8 An efficient realization of a two-bit adder

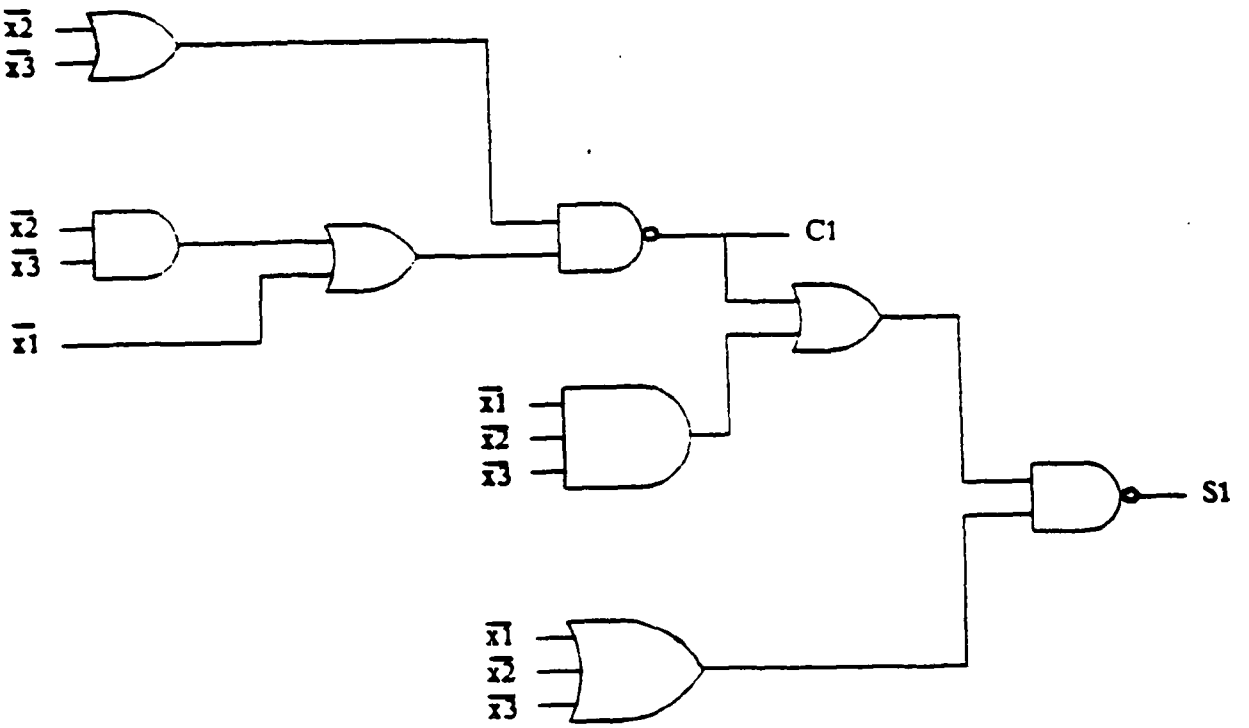


Figure 5.9 An alternate realization of a two-bit adder

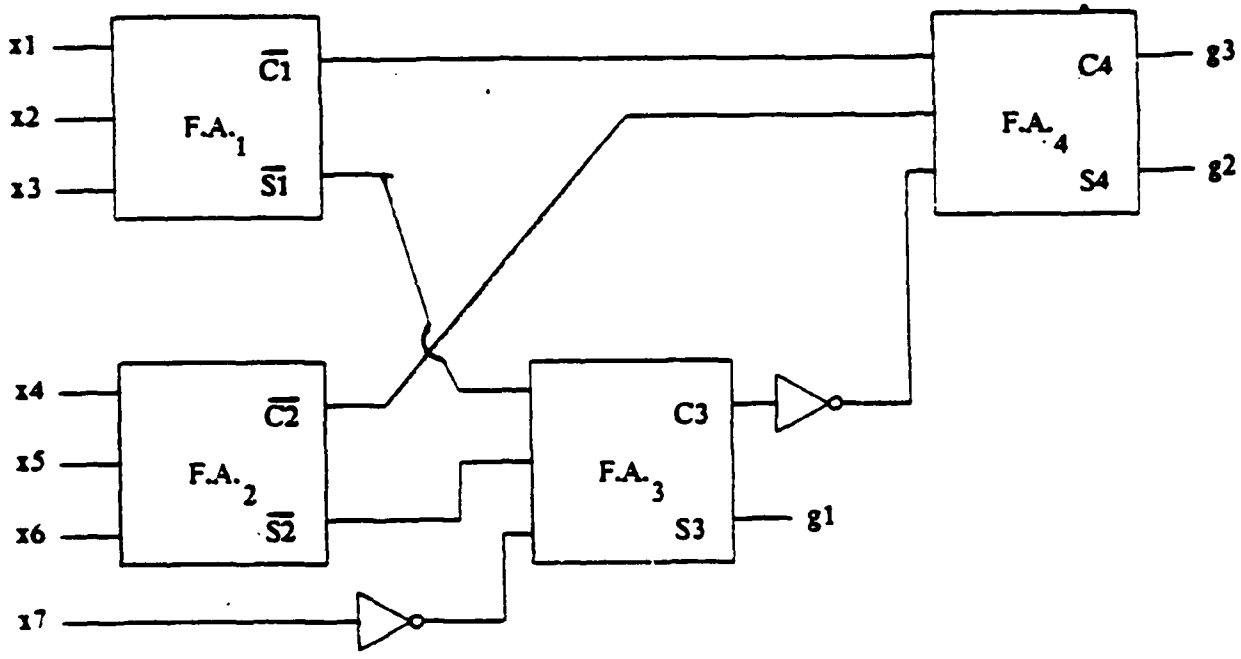


Figure 5.10 An efficient checkbit complement generator for $n_I = 7$

code checker:

$$n_T = 14(n_l - n_k) + 10(n_k - 1) + m = 14n_l - 4n_k - 10 + m \qquad (5)$$

where m is a constant required to account for some inverters that remain in the circuit. For Figure 5.10, m = 4, because two inverters are present in the circuit.

For the Berger code checker considered above, with $n_l = 7$ and $n_k = 3$, we get $n_T = 112$ from (4). If our techniques are used, however, we get $n_T = 80$, from (5). Hence we have reduced the transistor count by about 29%. This is roughly the reduction to be expected for most Berger code checkers.

Of course, the added advantage of speed-up of the circuit due to the removal of the inverters is also obtained by using our technique.

5.4. Cost reduction of MOS m-out-of-n code checkers

We showed in Section 5.2 that the three techniques presented there are individually or collectively applicable to all the known designs of m-out-of-n code checkers. We will present some tables below showing the reduction in the checker cost in the nMOS implementations of some of these checkers. L1 and C1 refer to the number of load and control transistors respectively if the m-out-of-n code checker designs were to be implemented directly. L2 and C2 refer to the number of load and control transistors respectively after the techniques in Section 5.2 are employed.

$$\% \text{ reduction in transistor count} = \frac{(L1 + C1 - L2 - C2)}{(L1 + C1)} * 100$$

For simplicity the constraints due to AND fan-in and the OR fan-in have not been taken into account.

Table 5.3 Transistor counts for Smith's m-out-of-2m code checkers

Code	L1	C1	L2	C2	% Reduction
3/6	16	32	2	24	46
4/8	36	66	10	48	43
5/10	64	112	12	80	48
10/20	324	522	82	360	48

Table 5.4 Transistor counts for Reddy's m-out-of-2m code checkers

Code	L1	C1	L2	C2	% Reduction
3/6	12	28	2	22	40
4/8	24	54	2	52	31
5/10	40	88	10	90	22
10/20	180	378	74	364	22

Table 5.5 Transistor counts for Anderson's m-out-of-2m code checkers

Code	L1	C1	L2	C2	% Reduction
3/6	4	26	2	22	20
4/8	4	66	2	48	29
5/10	4	162	2	98	40
10/20	4	10242	2	1176	89

Table 5.6 Transistor counts for Marouf-Friedman's m-out-of-n code checkers

Code	L1	C1	L2	C2	% Reduction
2/5	20	38	10	28	34
3/7	20	63	10	48	30
3/8	24	84	12	64	29
3/10	24	116	12	86	30
4/9	20	126	10	84	31
5/11	20	240	10	137	43
10/21	20	16346	10	1464	91

Table 5.7 Transistor counts for Piestrak's m-out-of-n code checkers

Code	L1	C1	L2	C2	% Reduction
3/7	26	52	15	40	30
3/8	30	62	16	46	33
3/10	42	84	26	66	27
4/9	38	83	25	67	24
5/11	52	116	35	94	23
10/21	132	383	95	326	18

Table 5.8 Transistor counts for Gaitanis-Halatsis' m-out-of-n code checkers

Code	L1	C1	L2	C2	% Reduction
2/5	24	42	15	33	27
3/7	32	67	17	48	34
3/8	32	75	18	56	31
3/10	36	102	20	76	30
4/9	38	98	19	73	32
5/11	42	151	21	107	34
10/21	62	1020	31	625	40

Looking at the above tables, we can conclude that for m-out-of-2m codes for small m, Anderson's designs [11] are best from the point of view of reduced transistor count; while for m-out-of-2m codes for large m, one could choose either Smith's [15] or Reddy's designs [40]. It is important to note that one should not equate a load transistor with a control transistor, because presence of a load transistor has the indirect effect of introducing extra routing for interconnections in the layout of the circuit. Hence it is difficult to decide whether Smith's or Reddy's design is better for larger m-out-of-2m codes until the actual layout is done for each individual case.

For smaller m-out-of-n ($n \neq 2m$) codes there is not much of a difference between the three designs shown above. For larger m-out-of-n codes Piestrak's [43] design seems to be the best from the chip area considerations. We have not given tables for the other two known designs for m-out-of-n codes, namely Nanya-Tohma's design [45] and Efstathiou-Halatsis' design [41], because these designs require much higher transistor count compared to the three designs for which tables were presented here. Hence they are not of much practical interest from the from the point of view of MOS implementation.

CHAPTER 6

DESIGN OF TSC EMBEDDED CHECKERS

6.1. Introduction

Frequently the functional circuit does not provide the checker with all the input vectors that the checker needs to be fully tested. Then it becomes necessary to have direct control over the input lines of such embedded checkers so that test patterns may be applied to fully test it. This, however, requires extra pins and/or circuitry on the chip and increases the complexity of the design.

This problem was considered by Anderson [11] and Smith [15] for building a self-checking network from self-checking blocks. Anderson required that each block be 'fully exercised'. This means that each block should receive all members of its input code space with the application of codewords to the primary inputs of the network. This is a very strong condition and is difficult to meet. Smith suggested that the blocks just need to be 'sufficiently exercised' in order to be self-testing. This means that they only need receive the required test patterns during the course of normal operation of the network.

In practice it is difficult to ensure that the blocks will be 'fully exercised' or even 'sufficiently exercised'. Khakbaz gave a set of sufficient conditions in [46] for the existence of 'sufficiently exercised' embedded parity checkers. A parity checker, however, can only detect single errors in its inputs and not unidirectional errors. He also extends his results to two-rail code checkers.

In this chapter we present a new encoding technique to encode the uncoded information part of the outputs of the functional circuit into a separable code, which can detect unidirectional errors. This encoding enables the embedded checker based on it to

be self-testing and, hence, TSC. Thus the problem of the embedded checker not being 'sufficiently exercised' is solved.

In [47] Wang et al gave a procedure for designing TSC circuits using PLAs. They require outputs of the PLA to form a code for which a TSC checker exists. We will take advantage of the approach given in [47] in the design of our TSC embedded checker. The enclosed portion in Figure 6.1 shows the general structure of our embedded checker. Let there be n information bits and k checkbits in the separable code. The information bits are fed to the PLA whose outputs form a k-variable two-rail code. A TSC checker for this two-rail code is connected to the PLA outputs. A bi-phase clock enables one of the rails of the two-rail code checker to be alternately connected to the checkbit lines and then to those outputs of the PLA which form one of the rails. So the two-rail code checker serves a dual purpose. The number of input vectors required to make our embedded checker self-testing is only 2^k.

6.2. Fault types of the PLA

The PLA has a structure like a ROM. It can be modeled as a two-level AND-OR array and can be implemented by NOR-NOR, NAND-NAND or AND-OR logic. The AND array forms the product terms and the OR array forms outputs which depend on activated product terms.

Some of the other properties of a PLA are, (a) n-bit addresses select only a fraction of 2^n words, (b) "don't care" condition is allowed in the address, (c) two or more words may be concurrently accessed from a single address.

The fault-types in a PLA and their effects that we will consider in this chapter have been taken from [47]. These are based on studies of fault-detection in PLAs [48-50].

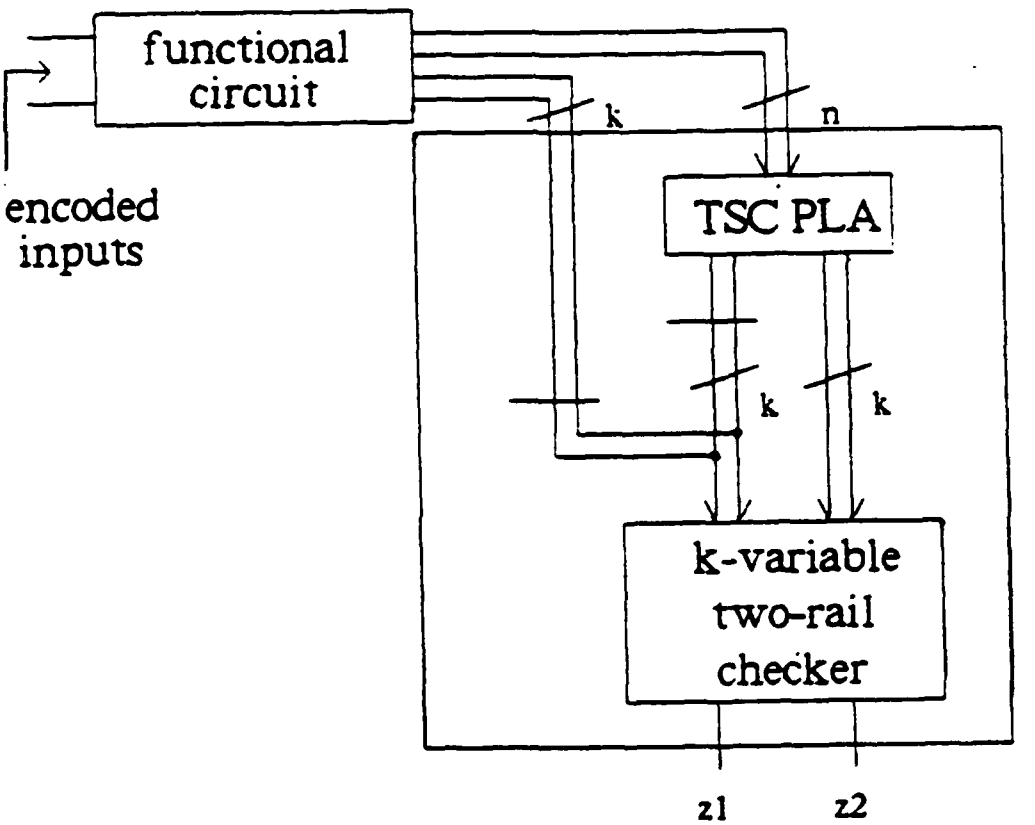


Figure 6.1 General structure of the embedded checker

They are (a) 'stuck' type fault, (b) crosspoint defect (extra device or missing device), and (c) shorts between adjacent lines.

The effects of these three types of faults are as follows:

(1) Faults in the AND array and in the input decoder may cause either the activation of unaddressed product terms, or the deactivation of addressed product terms, or do both, or simply have no effect.

(2) Faults in the OR array and in the output inverters (if used) may cause either the increment of 1's or the decrement of 1's in the output pattern, or both, or they may have no effect.

The above is applicable to NOR-NOR, NAND-NAND or AND-OR implementations of the PLA.

The fault-set for our checker will consist of all single faults from the above three fault-types in the PLA and all single stuck-at faults in the two-rail code checker.

6.3. A new encoding technique

This section describes a new encoding technique to encode a set of uncoded vectors and produce a set of separable coded vectors. If there are n bits in the uncoded vector and if k checkbits are required to encode it then our technique ensures that each of the 2^k bit-combinations forms the checkbit part of at least one coded vector. We will see later that this property is required to make our embedded checker self-testing.

We will use some notations and definitions given by Bose et al in [51].

If X and Y are two n-tuples over GF(2)=$\{0,1\}$ then we denote the number of 1-to-0 crossovers from X to Y by N(X,Y). For example, if X=(0010) and Y=(1100) then

N(X,Y)=1 and N(Y,X)=2.

Note that the two vectors X and Y are unordered iff $N(X,Y) \geq 1$ and $N(Y,X) \geq 1$.

Presently Berger encoding [52] is a standard method of obtaining a separable code from a set of uncoded vectors. As mentioned earlier, in this encoding the number of 0's in the uncoded vector is counted and binary representation of this number is concatenated as checkbits to this uncoded vector. The uncoded vector, which constitutes the information part, together with the checkbits concatenated to it, form the coded vector. If there are n information bits then $\lceil \log_2(n+1) \rceil$ checkbits are required. This encoding detects all unidirectional errors.

In [15] Smith suggested a modification to the above method of encoding to save on the number of checkbits. If the number of vectors is less than 2^n then in most cases the required number of checkbits will be less than $\lceil \log_2(n+1) \rceil$. Smith considers each member of the uncoded vector space one at a time and puts the vectors with the same number of 0's in one group. If number of such groups is m then he shows that only $\lceil \log_2 m \rceil$ checkbits are required for the encoding. Note that $m \leq (n+1)$.

We will now present an encoding technique which requires the same number of checkbits as Smith's encoding and also ensures the presence of all the 2^k possible combinations on the k checkbit lines.

We start with a set of uncoded vectors which will later form the information part of the coded vectors. We examine each vector, and if the number of 0's in the vector is r, we say r is its group number. If the number of bits in each vector is n then there can be at most $(n+1)$ groups. Let the actual number of distinct groups be m. Then, according to

Smith, only $k=\lceil \log_2 m \rceil$ checkbits are required to encode each vector. As mentioned before, for the success of our technique we need the presence of at least 2^k vectors. This is hardly a difficult condition to satisfy. For example, if the uncoded vectors have 7 information bits then our coding technique will require at most 3 checkbits for each uncoded vector, and presence of only 8 out of 128 possible uncoded vectors suffices to make our checker TSC.

Now we are faced with the problem of concatenating each of the 2^k bit-combinations, as checkbits, to some uncoded vector(s) so that we finally get a set of coded vectors which form an unordered code.

If $m=2^k$ then the solution is straightforward. The k-bit combination with the highest value (111..11) is concatenated to vector(s) with the highest group number. The next highest k-bit combination (111..10) is concatenated to vectors with the next highest group number and so on.

If $m<2^k$ then let $2^k-m=p$. Now within the same group all uncoded vectors cannot be concatenated with the same k-bit combination as checkbits. The process of concatenating checkbits to uncoded vectors is still similar to the $m=2^k$ case, i.e., the highest k-bit combination is concatenated to vectors with the highest group number and so on. But now we will allow vectors in the same group to have different k-bit combinations concatenated to them as checkbits. This way the extra k-bit combinations, p in number, can be taken care of. But any two vectors from different groups are still not allowed to have the same k-bit combination as checkbits. That this way of concatenating checkbits will result in an unordered code will be proved later in a theorem.

Let us formalize the above paragraphs with a step-by-step procedure. We assume that the number of vectors in the uncoded vector space is greater than or equal to 2^k where k is the number of checkbits required. Hence since there are at least 2^k vectors present, each of the k-bit combinations is assured to be concatenated to at least one vector.

Procedure 6.1

(1) Consider one uncoded vector from the uncoded vector space at a time. If the vector has r 0's, it will be said to have group number r.

(2) If the number of distinct groups is m then the number of checkbits required for each vector is $k = \lceil \log_2 m \rceil$.

(3) Concatenate k-bit combinations as checkbits to each uncoded vector so that the following conditions are satisfied:

(a) if the highest group number is s, the k-bit combination with highest value (111..11) is concatenated to vector(s) with the group number s. The next higher k-bit combination (111..10) can be concatenated to vector(s) with the group number s, if there are any unconcatenated vectors left in that group, or to vector(s) with group number (s-1) and so on.

(b) A k-bit combination cannot be assigned until all combinations with higher value have already been assigned.

(c) Each of the 2^k k-bit combinations is assigned to at least one vector.

(d) A k-bit combination is not assigned to vectors with different group numbers.

The following example will make things clear:

Example 6.1: Let the set of uncoded vectors be {0000,0001,0010,0111,1011,1110}. Table 6.1 shows the encoding.

Table 6.1 An example showing our encoding technique

uncoded vector	group number	first encoding	second encoding
0000	4	000011	000011
0001	3	000110	000110
0010	3	001010	001001
0111	1	011101	011100
1011	1	101101	101100
1110	1	111000	111000

Since there are vectors belonging to three groups, m=3 and $k = \lceil \log_2 m \rceil = 2$. Hence two checkbits are required for each vector. But $2^k - m = 1$. So there is one group of vectors all of whose members do not have the same checkbits. This is seen to be true for vectors with group number 1 for the first encoding and for vectors with group number 3 for the second encoding. It is clear that various other valid encodings are also possible.

Theorem 6.1: Procedure 6.1 produces a code which is capable of detecting all unidirectional errors.

Proof: We will take advantage of Theorem 5 given with proof in [51]. This states:

A code C is capable of detecting all unidirectional errors iff every pair of codewords is unordered, i.e.,

for all distinct $X_1, X_2 \epsilon C$

$$N(X_1, X_2) \geq 1 \text{ and } N(X_2, X_1) \geq 1$$

When checkbits are assigned to an uncoded vector by Procedure 6.1 a coded vector is formed. All such coded vectors form the code space. Thus every coded vector has two separable parts, an n-bit information part and a k-bit checkbit part. We will refer to these two parts of a vector X as V and W respectively.

Let us take any two vectors X_1 and X_2 from the code space. Two cases can arise and we will treat them separately.

Case 1: The information parts (uncoded parts) V_1 and V_2 of vectors X_1 and X_2 respectively have the same group number.

This means V_1 and V_2 have the same number of 0's. Since V_1 and V_2 are necessarily distinct, the only way they can have the same number of 0's is if there is at least one bit-position in V_1 which has a 1 where the corresponding bit-position in V_2 has a 0 and vice-versa. This implies:

$$N(V_1, V_2) \geq 1 \text{ and } N(V_2, V_1) \geq 1 \qquad (1)$$

$$\text{But } N(X_1, X_2) = N(V_1, V_2) + N(W_1, W_2) \qquad (2)$$

$$\text{and } N(X_2, X_1) = N(V_2, V_1) + N(W_2, W_1) \qquad (3)$$

So from (1), (2) and (3) it follows that $N(X_1, X_2) \geq 1$ and $N(X_2, X_1) \geq 1$.

Hence X_1 and X_2 form an unordered pair. Thus if the information parts of any two coded vectors have the same group number then the coded vectors form an unordered pair.

Case 2: The information parts V_1 and V_2 of vectors X_1 and X_2 respectively have different

group numbers.

Let the group number of V_1 be r_1 and that of V_2 be r_2. Without loss of generality let us assume $r_1 > r_2$. This means V_1 has more 0's than V_2. Hence there is at least one bit-position in which V_2 has a 1 and V_1 has a 0. This implies that $N(V_2, V_1) \geq 1$ and therefore from equation (3)

$$N(X_2, X_1) \geq 1 \qquad (4)$$

Since $r_1 > r_2$, the steps 3(a), 3(b) and 3(c) of Procedure 6.1 ensure that W_1 has a higher value than W_2. It is a property of any two binary vectors Y_1 and Y_2 that if value of Y_1 is greater than value of Y_2 then $N(Y_1, Y_2) \geq 1$. Hence in the above case $N(W_1, W_2) \geq 1$. Equation (2) then implies that

$$N(X_1, X_2) \geq 1 \qquad (5)$$

From (4) and (5) it follows that X_1 and X_2 form an unordered pair.

Combining the results of Case 1 and Case 2 we can say that every pair of coded vectors from the code space forms an unordered pair. Hence, according to Theorem 5 in [51], the code space is capable of detecting all unidirectional errors.

□

6.4. Design procedure for the TSC embedded checker

We will depend heavily on the design procedure given in [47] for TSC circuits using PLA's. In Section 4 of [47] the following conditions are imposed on a PLA to make it TSC:

(1) One and only one product term is activated by each input pattern, that is, the PLA is nonconcurrent.

(2) The output pattern belongs to a checkable code, i.e., a code for which a TSC checker exists.

In our design the n information bits of the coded vectors are fed to a PLA as inputs, and the outputs of the PLA form a two-rail code. One rail of this code is the same as checkbits for those information bits and the other rail is the complement of these checkbits. For any input to the PLA which does not form the information part of any coded vector the output of the PLA is a non-code.

After applying Steps 1 through 4 of Section 4 in [47] two cases may arise as follows:

Case 1: No extra line is required at the output of the PLA. In this case we use the configuration given in Figure 6.1. We will prove later that this configuration is TSC.

Case 2: One extra line y_k is required at the output of the PLA to remove the concurrency in Step 3 and make shorts between adjacent lines testable in Step 4. In this case the configuration of Figure 6.2 is used. In general the line y_k has a fixed value (1 or 0) only for a few product term lines, values for other product term lines being "don't cares". Another line c_k is added to the output of the PLA and it has a 1 when y_k has a 0 and vice-versa. Now the 2-tuple (0,1) or (1,0) can be assigned to (y_k,c_k) on the "don't care" product term lines in such a way that the 2-variable two-rail code checker gets all the members of its input code space {0101,0110,1001,1010}. This is hardly a difficult condition to satisfy. One can come up with pathological cases in which it is not possible to supply all members of its code space to the 2-variable two-rail code checker, in which case our design fails. But these cases are extremely rare indeed.

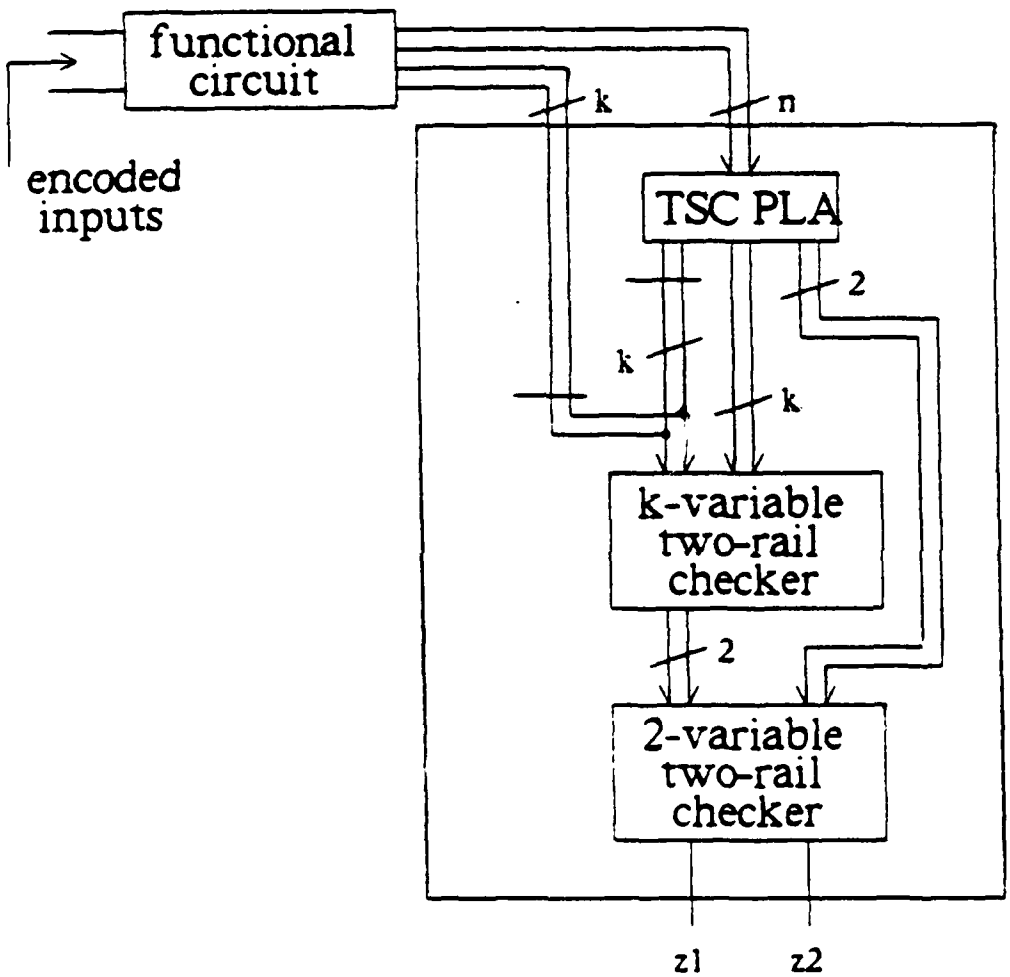


Figure 6.2 Modified structure of the embedded checker

To show that the embedded checker obtained by connecting the PLA and the two-rail code checker(s), as in Figures 6.1 or 6.2, is TSC, we will show that it is (a) code-disjoint, and (b)self-testing. The fault-set is given in Section 6.2. Only Figure 6.2 will be considered because the same arguments are applicable to Figure 6.1.

A known TSC design is used for the two-rail code checkers. Since both the two-rail code checkers are fully exercised they are indeed TSC.

(a) *Code disjoint*

If a non-code word appears at the input to the embedded checker it means there is an error in the information part, or the checkbit part of the coded vector, or both.

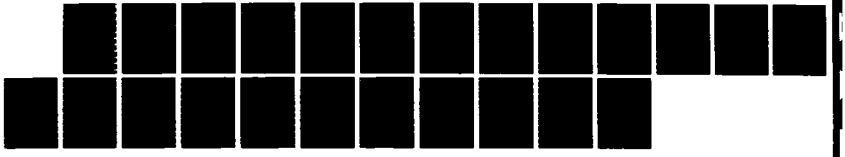
If the error has changed the information part of the intended vector to the information part of another vector, it follows from the fault-secure property of the functional circuit that the checkbits will be wrong for the received vector. Hence a non-code will appear on Phase 2 at the inputs of the k-variable two-rail code checker. Since both the two-rail code checkers are code-disjoint, their non-code inputs will be reflected at their outputs as non-codes as well. Hence a non-code word will appear on y_1 and y_2 on Phase 2.

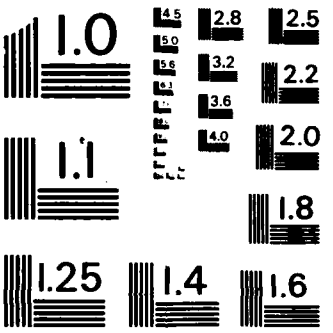
If the error has changed the information part of the intended vector such that it does not form the information part of any other coded vector, the output of the PLA will be a non-code word and hence this will be detectable, by previous arguments, at y_1 and y_2 on Phase 1.

Similarly, if only the checkbit part has an error then y_1 and y_2 will have a non-code word on Phase 2 because the two rails to the k-variable two-rail checker will not have complementary inputs.

AD-A169 756 TOTALLY SELF-CHECKING CIRCUITS AND TESTABLE CMOS CIRCUITS(U) ILLINOIS UNIV AT URBANA COMPUTER SYSTEMS GROUP N K JHA JUN 86 CSG-51 N00039-80-C-0556 2/2

UNCLASSIFIED F/G 9/5 NL





(b) *Self-testing*

If there is a fault in the PLA then either the k-variable or the 1-variable or both set of two-rail lines will have a non-code on them for at least one of the inputs to the embedded checker. This is detected on Phase 1 at y_1 and y_2. If there is a fault in either of the two-rail code checkers then their inputs are known to be codes and at least one of these codes will produce a non-code at the checker's output which will be reflected as a non-code at y_1 and y_2 on both Phase 1 and Phase 2.

CHAPTER 7

TESTABLE CMOS LOGIC CIRCUITS

7.1. Introduction

It was mentioned in Chapter 4 that most of the existing methods of generating test sets for a CMOS logic circuit are based on a zero delay assumption through all gates and paths. When arbitrary delays are assumed, the test sets derived by these methods can be shown to be invalidated [20]. In [20] Reddy et al also gave some examples to point out that, (1) there exist certain functions for which an AND-OR (NAND-NAND) or an OR-AND (NOR-NOR) irredundant CMOS realization does not have any valid test set under arbitrary delays, (2) if an undetectable stuck-open fault is present, it may invalidate other tests or the circuit may malfunction for some other pair of input changes. They follow the approach of designing a testable CMOS circuit for a given function by using extra controllable inputs and additional logic.

For example, the AND-OR CMOS realization of the function f in Figure 7.1 does not have a valid test set in the presence of arbitrary delays.

In [53-54] methods were given to derive test sets for stuck-open faults, which remain valid under arbitrary delays. Their usefulness is limited, however, to CMOS realizations for which such a test set exists.

In this chapter, we present a necessary and sufficient condition to find out if there exists a valid test set, under arbitrary delays, for the irredundant AND-OR or OR-AND CMOS realizations, for any given function. We show that the test set can be derived easily, if one exists. We use the Hybrid CMOS realization, introduced earlier, to take care of the rare cases in which a valid test set does not exist for an AND-OR or OR-AND

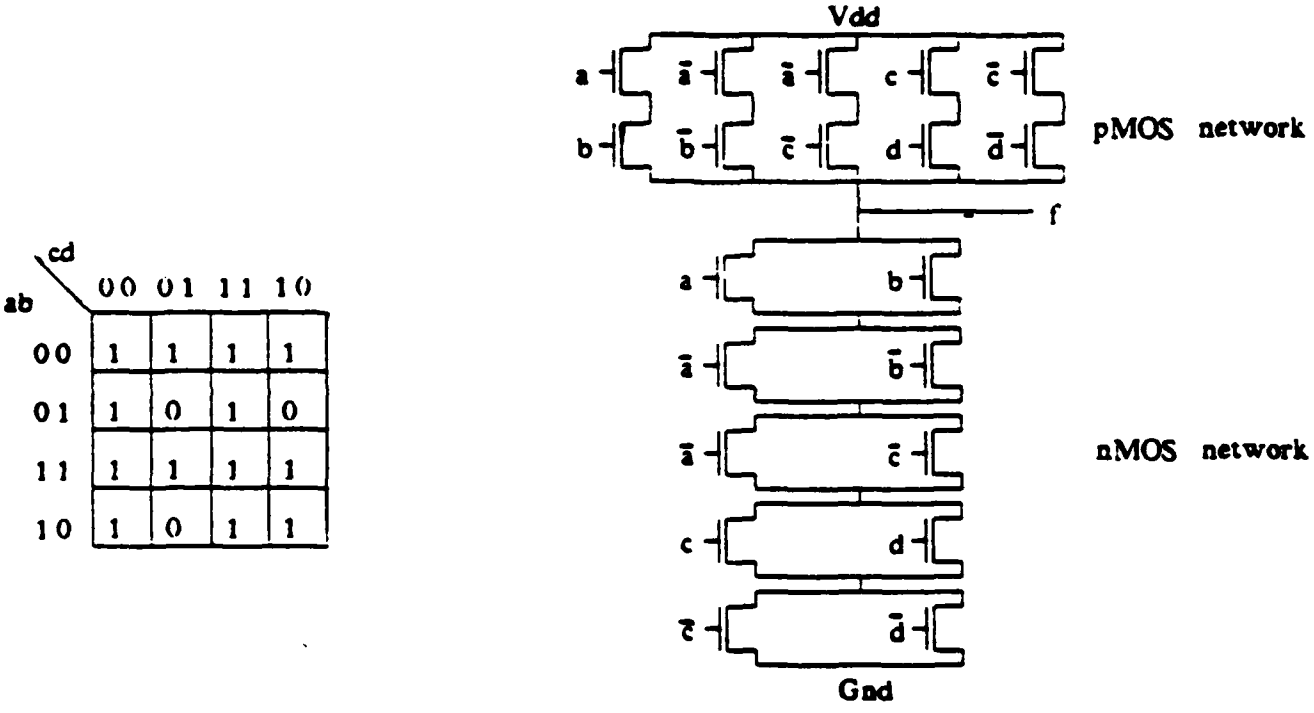


Figure 7.1 An AND-OR CMOS realization of a function f

CMOS realization. We show that a valid test set is guaranteed to exist for this realization. We do not require any extra controllable inputs or logic for making this realization testable.

7.2. Definitions

Definition 7.1: A true-vertex (false-vertex) of a function is the input for which the function is 1 (0).

Definition 7.2: The set of true-vertices (false-vertices) corresponding to a prime implicant (implicate) is said to belong to a prime implicant (implicate) loop.

Definition 7.3: Considering the prime implicant (implicate) loops corresponding to an irredundant sum of products (product of sums) expression of any function, if a true-vertex (false-vertex) belonging to one loop does not also belong to any other loop, then it is said to be a 'special' true-vertex (false-vertex), otherwise it is said to be an 'ordinary' true-vertex (false-vertex).

In the Karnaugh map in Figure 7.1, the true-vertex (1010) is a special true-vertex for all irredundant sum of products expressions of this function.

Definition 7.4: The distance D(X,Y) between two vectors X and Y is the number of bit-positions in which they differ.

For example, D(1101, 0111) = 2.

Definition 7.5: An operation R is defined on two vectors X and Y as, R:(X,Y) $\rightarrow$ (V,W), where V and W are, respectively, the bit-wise AND and bit-wise OR of X and Y.

For example, R:(0110,1010) → (0010,1110).

7.3. Conditions for testability

Lemma 7.1: In a test set for testing stuck-open faults in a CMOS realization of any function, if for every initialization-test input pair (X,Y), D(X,Y) = 1, then the test set cannot be invalidated in the presence of arbitrary delays in the gate.

Proof: If D(X,Y) = 1, then there can be no intermediate vectors generated in the transition from X to Y, and, hence, no spurious conduction paths can be activated in the CMOS complex gate. This implies that arbitrary delays cannot invalidate the test set.

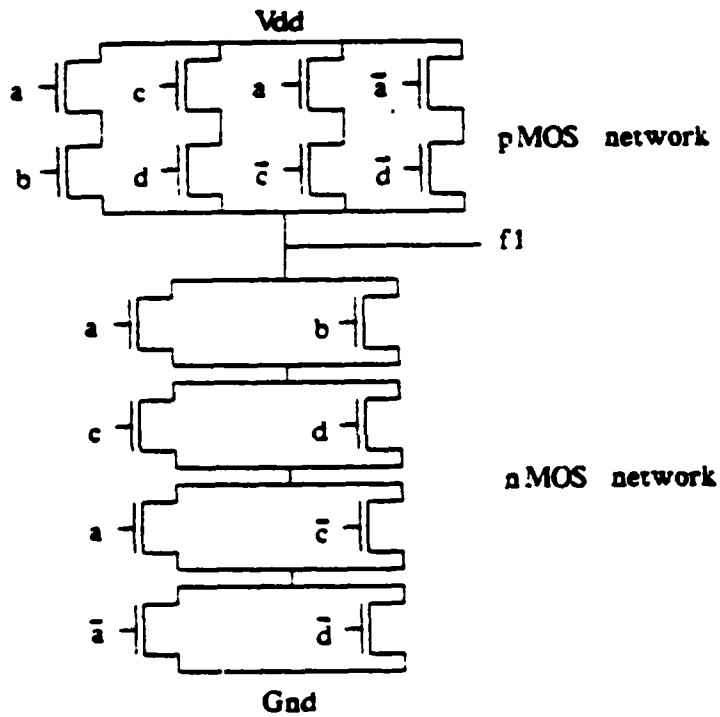
□

One might have thought that it would be sufficient to have the initialization-test input pairs form an ordered pair. But the following example will show that this is not so.

Example 7.1: Consider the CMOS circuit in Figure 7.2 and its accompanying test set.

The initialization-test pair formed by Row 9 and Row 10 is supposed to check any stuck-open fault in the conduction path a-c-a-$\bar{d}$ in the nMOS network. But two transitions are possible from Row 9 to Row 10, namely (1111-1011-1010) and (1111-1110-1010). The first transition does not invalidate the test set, but the second one does. The intermediate vector 1110 activates a spurious conduction path b-c-a-$\bar{d}$. Hence a stuck-open fault in the first member of the intended path a-c-a-$\bar{d}$ is not detected.

It can easily be found that replacing the vector 1111 in Row 9 by 1011 solves this problem, since now the condition in Lemma 7.1 is satisfied. Note that by doing this,



	a	b	c	d	f1
1	0	1	0	1	0
2	0	0	0	1	1
3	0	1	0	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	1	1
7	0	1	0	1	0
8	1	1	0	1	1
9	1	1	1	1	1
10	1	0	1	0	0

Figure 7.2 A CMOS circuit and its test set

Row 8 and Row 9 are no longer distance 1 away. But this does not matter since Row 8 and Row 9 do not form an initialization-test pair.

The condition given in Lemma 7.1 is sufficient, but not necessary. In other words, there exist initialization-test pairs which have distance greater than 1 between them, or which are unordered, and yet form a valid two-pattern test in the presence of arbitrary delays. However, most test set invalidations occur because some initialization-test input pair forms an unordered pair.

The above statements imply that Lemma 7.1 gives us a conservative, yet provably valid, way of generating a test set under arbitrary delays.

Lemma 7.2: For two unordered vectors X and Y, if R:(X,Y) → (V,W), then both V and W may potentially appear as intermediate vectors in the transition from X to Y, under arbitrary delays.

Proof: From the properties of bitwise ANDing and bitwise ORing it is clear that both X and Y cover V and are covered by W. If the delays cause all the 1-to-0 transitions in X to occur before any 0-to-1 transition occurs, then V will appear as an intermediate vector. Similarly if all the 0-to-1 transitions in X occur before any 1-to-0 transition occurs, then W will appear as an intermediate vector.

□

Lemma 7.3: For two vectors X and Y, if R:(X,Y) → (V,W), then V and W can be true-vertices (false-vertices) belonging to one prime implicant (implicate) loop only if X and Y both are also true-vertices (false-vertices) of the same prime implicant (implicate) loop.

Proof: If X and Y are ordered and assuming, without loss of generality, that Y covers X, then V=X and W=Y, and the lemma is trivially true.

Even if X and Y are unordered, V and W always form an ordered pair, since in the bit-positions that V and W differ, V has 0's and W has 1's. For two ordered vectors V and W, such that W covers V, to be true-vertices (false-vertices) belonging to one prime implicant (implicate) loop it is necessary that every vector that covers V and is covered by W, also be a true-vertex (false-vertex) of the same prime implicant (implicate) loop. One can easily verify this with the aid of a Karnaugh map. From Definition 7.5, both X and Y cover V and are covered by W. Hence V and W can belong to one prime implicant (implicate) loop only if X and Y also do so.

□

Theorem 7.1: There does not exist a valid test set, under arbitrary delays, for an AND-OR (OR-AND) CMOS realization if and only if at least for one prime implicant (implicate) loop all its special true-vertices (false-vertices) are distance greater than 1 away from any false-vertex (true-vertex).

Proof: We will consider the AND-OR CMOS realization only. Dual arguments can be applied to the OR-AND CMOS realization. We will prove the "if" part of the Theorem first and then the "only if" part.

"*If*": Every prime implicant loop corresponding to an irredundant AND-OR CMOS

realization has at least one special true-vertex, otherwise its presence will make the circuit redundant. Let P be a prime implicant loop, all of whose special true-vertice are distance greater than 1 away from any false-vertex. Every prime implicant has a corresponding series conduction path in the pMOS network of the AND-OR CMOS realization; the inputs to the transistors in this path being the complement of the variables constituting the prime implicant. To test any transistor in this path for a stuck-open fault one has to apply a false-vertex as an initializing input, and follow it with a special true-vertex belonging to the corresponding prime implicant loop as a test input. Let these initializing and test inputs to test the conduction path corresponding to P be X and Y respectively. Note that if we use an ordinary true-vertex as a test input then conduction path(s) corresponding to the other prime implicant loop(s) to which that ordinary true-vertex belongs will also be activated, thus invalidating the test set. Let us now consider two different cases separately.

Case 1: X and Y are ordered and $D(X,Y) \geq 2$

It is easy to see that if X (Y) covers Y (X) then there exists a vector Z such that X (Y) covers Z and Z covers Y (X), and $D(X,Z) = 1$. Z can be one of three things, (1) an ordinary true-vertex of P, (2) true-vertex of some other prime implicant, (3) a false-vertex. For (1) and (2) the test set is invalidated because Z can potentially occur as an intermediate vector in the transition from X to Y. For (3) D(Y,Z) has to be ≥ 2, otherwise the condition in this theorem is violated. Hence there still are other intermediate vectors only distance 1 away from Y, which can occur in the transition from Z to Y. This implies that ultimately (1) or (2) has to be satisfied for a transition from Z to Y, thus invalidating the test set.

Case 2: X and Y are unordered

Let R:(X,Y) → (V,W). From Lemma 7.2 both V and W can potentially occur as intermediate vectors in the transition from X to Y under arbitrary delays. If V and W are both true-vertices, then from Lemma 7.3, V and W cannot belong to the same prime implicant loop, since X, being a false-vertex, can never belong to a prime implicant loop. So even if one of V or W belonged to P, the other would not. So at least one other prime implicant can potentially be activated, thus invalidating the test set.

If V or W or both are false-vertices, then without loss of generality, let us assume that V is a false-vertex. Then from the condition in the theorem it follows that D(V,Y) $\geq$ 2. Since V and Y are always ordered, this is the same as Case 1, if V is substituted for X and the new values of V and W are computed.

"*Only if*": By extending arguments used in the proof of Theorem 3 [20], it can easily be shown that valid two-pattern tests, under arbitrary delays, exist to test each stuck-open fault in the nMOS network of the AND-OR CMOS realization. We will extend t' .e arguments here for completeness. Corresponding to every prime implicant loop from an irredundant AND-OR CMOS realization, there exists a set of transistors connected in parallel in the nMOS network. When any true-vertex from this prime implicant loop is applied to the circuit, the inputs to all the transistors in this set become logic 0. Hence the output node assumes the logic value 1, and this true-vertex acts as an initialization input. Now in order to detect a stuck-open fault in a transistor from this set, a test input is applied, which makes the input to this transistor logic 1, while retaining logic 0 at the inputs to other transistors in that set. This test input, of course, causes a logic 0 to appear at the output node, in the fault-free case. Note that there always exists a test

input like the one specified above, otherwise the transistor in question must be redundant. From Lemma 7.1, test invalidation cannot occur for this initialization-test input pair.

Trivially extending this argument further, all the stuck-open faults in the nMOS network are seen to be detectable.

If at least one special true-vertex of every prime implicant loop is only distance 1 away from some false-vertex then, from Lemma 7.1, this false-vertex and the special true-vertex form a valid two-pattern test for testing the transistors in the series conduction path in the pMOS network of the realization, corresponding to that prime implicant. Hence, again, with a trivial extension of this argument, all the stuck-open faults in the pMOS network are seen to be detectable.

Combining all the two-pattern tests found for the nMOS and pMOS networks, we have a test set that cannot be invalidated in the presence of arbitrary delays.

□

We can reduce the size of the test set by recognizing that an initializing input for testing a stuck-open fault in the nMOS network is often same as the test input for testing another stuck-open fault in the pMOS network, and vice-versa. These vectors can be placed adjacent to each other in the test set, thus reducing its size.

Example 7.2: The function shown in Figure 7.1 satisfies the condition in Theorem 7.1 for its AND-OR CMOS realization. Hence for this realization there does not exist a valid test set under arbitrary delays.

We will now establish the testability of the Hybrid CMOS realization in Theorem

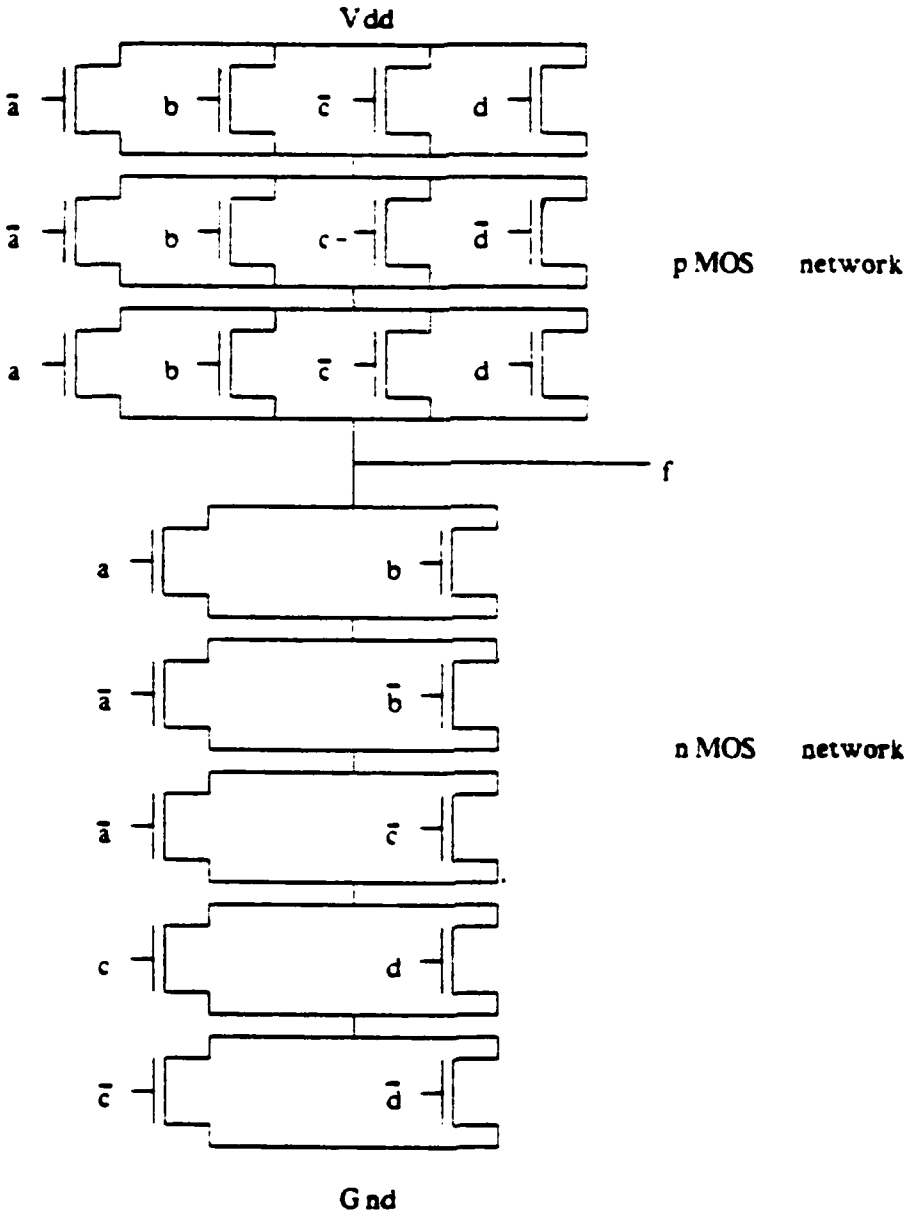


Figure 7.3 Hybrid CMOS realization of the function f

7.2. Figure 7.3 gives this realization for the function f given in Figure 7.1.

Theorem 7.2: There always exists a valid test set under arbitrary delays for a Hybrid CMOS realization of any function.

Proof: We have already seen in the "Only if" section of the proof of Theorem 7.1 that all the stuck-open faults in the nMOS network of an AND-OR CMOS realization are detectable by a test set that cannot be invalidated by arbitrary delays. This means that the same is true for the nMOS network of the Hybrid CMOS realization also, because it is the same as the nMOS network of the AND-OR CMOS realization. Now if we use exactly dual arguments for the OR-AND CMOS realization it can easily be shown that its pMOS network has a test set that cannot be invalidated by arbitrary delays. This again means that the pMOS network of the Hybrid CMOS realization is also testable in the presence of arbitrary delays.

□

We know, from Theorem 4.2, that the Reduced Hybrid CMOS realization will also have a test set that is guaranteed to be valid in the presence of arbitrary delays in the circuit. Figure 7.4 gives the Reduced Hybrid CMOS realization of the function f in Figure 7.1.

It is quite rare for a function to satisfy the condition in Theorem 7.1. We have only been able to find 16 functions of 4 variables, out of a possible 2^{16}, whose AND-OR CMOS realizations do not have valid test sets under arbitrary delays. For all these 16 functions it was found that the OR-AND CMOS realizations have valid test sets under arbitrary delays. Similarly, we found 16 other functions (inversions of the former) which do not

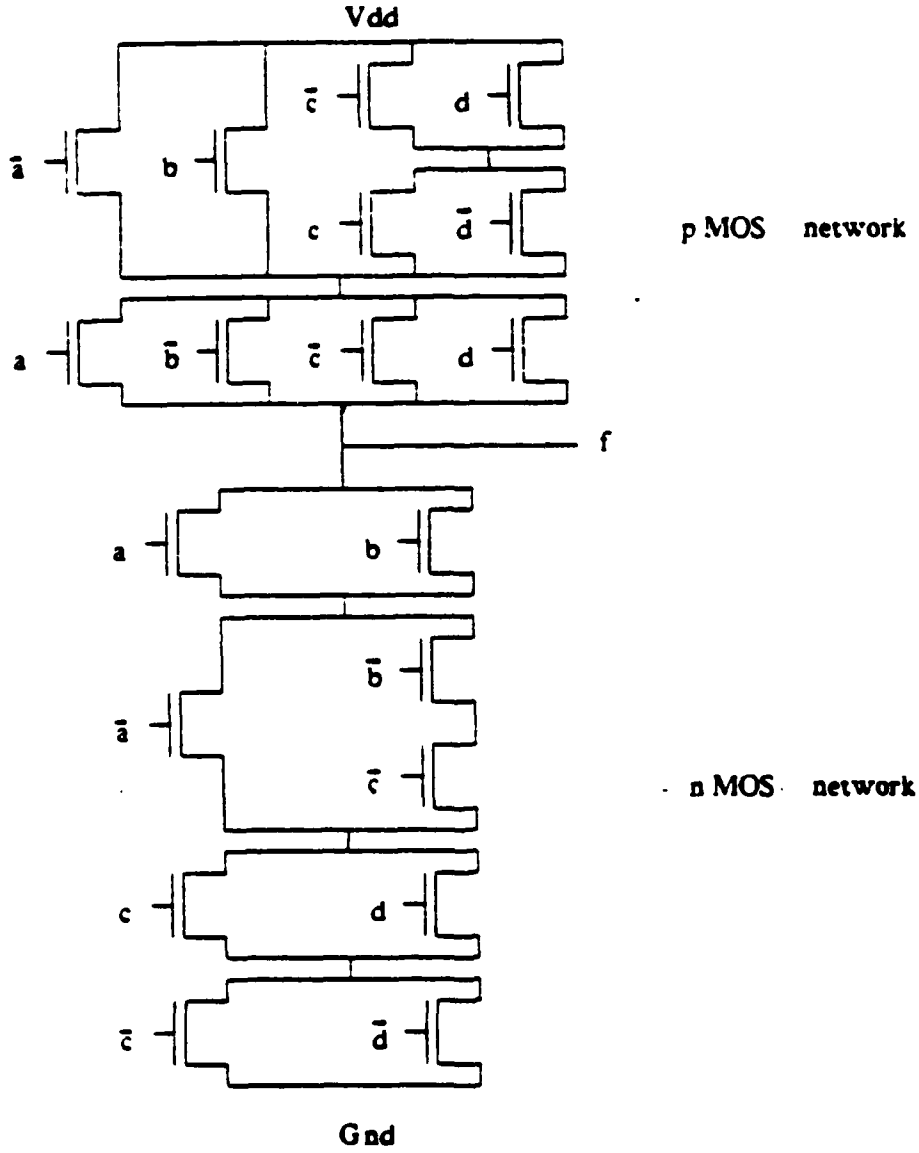


Figure 7.4 The Reduced Hybrid CMOS realization of the function f

have valid test sets for their OR-AND CMOS realizations. But again their AND-OR CMOS realizations had valid test sets. Only in extremely rare cases indeed will it happen that neither an AND-OR nor an OR-AND CMOS realization of the function has a valid test set. In such instances we use the Hybrid CMOS realization or its reduced version as mentioned earlier. Even when the AND-OR or OR-AND CMOS realizations have valid test sets, it might be worth implementing the function with a Reduced Hybrid CMOS realization, because sometimes this might require fewer transistors and tests.

CHAPTER 8

CONCLUSION

8.1. Concluding Remarks

The purpose of this thesis has been to investigate topics in the area of TSC MOS *circuits and testable CMOS circuits.*

The thesis began by introducing the concept of a TSC circuit and by providing a realistic physical failure model. Then implementation of TSC circuits in nMOS, Domino-CMOS and CMOS technnologies was discussed. Conditions and layout rules were given to make these circuits TSC with respect to realistic physical failures. A new Hybrid CMOS realization was introduced to enable detection of stuck-open faults in CMOS technology in the presence of arbitrary delays in the circuit. The stuck-on faults still pose a problem in CMOS technology in that they may or may not be detected by just monitoring logic levels. Monitoring the current in the circuit is required to detect such faults. Perhaps by imposing design restrictions and layout rules it might be possible to ensure the detection of such faults through the monitoring of logic levels. The dynamic CMOS technologies are more testable than the static CMOS technology, and their popularity should grow with the designers of ICs.

Techniques for reducing the transistor count, tests, and delay of TSC MOS checkers were presented next. Up to 90% reduction in the transistor count was obtained for some of these checkers. This results in even greater savings in chip area because of the additional reduction in the interconnections. The additional advantages were reduction in the test size and the delay of the circuit. It is apparent that the concepts that are applicable to the gate-level designs are not always applicable to their MOS

implementations. Hence one should look at the MOS implementations to evaluate their technological cost.

The problem of embedded TSC checkers was solved by providing a new encoding technique for realizing a separable code, and by using a design procedure based on this technique. This technique can be used when traditional encoding methods fail to provide all the codewords required to make the checker self-testing. Since, for most embedded checkers, such is generally the case, the usefulness of this technique is apparent. Designing efficient embedded checkers for non-separable codes is an area of future research.

The last problem to be discussed was concerned with making CMOS circuits testable in the presence of arbitrary delays. It has been shown that for some CMOS realizations a valid test set, under arbitrary delays, does not even exist. For such cases the new Hybrid CMOS realization or its reduced version can be used, for which a valid test set is guaranteed. The Hybrid CMOS realizations can also be used as an alternative to the AND-OR and OR-AND CMOS realizations, even when valid test sets exist for these realizations.

REFERENCES

[1] J. P. Roth, et al., "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," *IEEE Transactions on Computers,* Vol. C-16, pp. 567-579, Oct. 1967.

[2] W. C. Carter and P. R. Schneider, "Design of dynamically checked computers," *IFIP '68,* Edinburgh, Scotland, Vol. 2, pp. 878-883, Aug. 1968.

[3] D.A. Anderson, "Design of self-checking digital networks using coding techniques," Technical Report R-527, Coordinated Science Laboratory, University of Illinois, Sept. 1971.

[4] J. G. Tryon, "Quadded logic," in *Redundancy Techniques for Computing Systems,* Wilcox and Mann., eds., pp. 205-228, Spartan Books, Washington D. C., 1962.

[5] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Annals of Mathematical Studies,* No. 34, pp. 43-98, Princeton University Press, Princeton, N. J., 1956.

[6] A. Avizienis, "Design of fault-tolerant computers," *Proceedings of the Fall Joint Computer Conference,* pp. 733-743, 1967.

[7] R. W. Cook, et al., "Design of self-checking microprogram conntrols," *IEEE Transactions on Computers,* Vol. C-22, No. 3, pp. 255-262, Mar. 1973.

[8] M. Diaz and J. M. Desouza, "Design of self-checking microprogram controls," *5th International Symposium on Fault-Tolerant Computing,* Paris, France, pp. 137-142, June 1985.

[9] D. S. Ho, "The study of a totally self-checking adder," Coordinated Science Laboratory Report R-582, University of Illinois, Aug. 1972.

[10] M. Diaz, "Design of totally self-checking and fail-safe sequential machines," *4th International Symposium on Fault-Tolerant Computing,* Urbana, Illinois, pp. 3.19-3.24, June 1974.

[11] D. A. Anderson and G. Metze, "Design of totally self-checking check circuits for m-out-of-n codes," *IEEE Transactions on Computers,* Vol. C-22, pp. 263-269, Mar. 1973.

[12] W. C. Carter, K. A. Duke and D. C. Jessep, "A simple self-testing decoder checking circuit," *IEEE Transactions on Computers,* Vol. C-20, pp. 1413-1414, Nov. 1971.

[13] M.J. Ashajee and S.M. Reddy, "On Totally Self-Checking checkers for separable codes," *IEEE Transactions on Computers,* Vol. C-26, pp. 737-744, Aug. 1977.

[14] M.A. Marouf and A.D. Friedman, "Efficient design of self-checking checker for any m-out-of-n code," *IEEE Transactions on Computers,* Vol. C-27, pp. 482-490, June 1978.

[15] J. E. Smith," The design of Totally Self-Checking combinational circuits," Ph.D. thesis, Report R-737, Univ. of Illinois, Urbana, Illinois, Aug. 1976.

[16] Y. Crouzet and C. Landrault, "Design of Self-Checking MOS-LSI Circuits: Application to a four-bit Microprocessor," *IEEE Transactions on Computers* Vol. C-29, pp. 532-537, June 1980.

[17] S.R. Manthani and S.M. Reddy, "On CMOS totally self-checking circuits," *International Test Conference,* Philadelphia, pp. 866-877, Oct. 1984.

[18] J. Galiay, Y. Crouzet and M. Vergniault, "Physical versus logical fault models MOS LSI circuits: impact on their testability," *IEEE Transactions on Computers,* Vol. C-29, pp. 527-531, June 1980.

[19] R. L. Wadsack," Fault modelling and logic simulation of CMOS and MOS integrated circuits," *Bell System Technical Journal,* Vol. 57, pp. 1449-1474, May-June 1978.

[20] S. M. Reddy, M. K. Reddy and J. G. Kuhl, "On testable design for CMOS logic circuits," *International Test Conference,* Philadelphia, pp. 435-445, Oct. 1983.

[21] N. K. Jha and J. A. Abraham, "Totally self-checking MOS circuits under realistic physical failures," *International Conference on Computer Design,* Port Chester, New York, Oct. 1984.

[22] N. K. Jha and J. A. Abraham, "Totally self-checking CMOS circuits using a Hybrid realization," *15th International Symposium on Fault-Tolerant Computing,* Michigan, Ann Arbor, June 1985.

[23] N. K. Jha and J. A. Abraham, "Techniques for efficent MOS implementation of totally self-checking checkers," *15th International Symposium on Fault-Tolerant Computing,* Michigan, Ann Arbor, June 1985.

[24] N. K. Jha and J. A. Abraham, "The design of totally self-checking embedded checkers," *14th International Symposium on Fault-Tolerant Computing,* Orlando, Florida, pp. 265-270, June 1984.

[25] N.K. Jha and J.A. Abraham, "Testable CMOS logic circuits under dynamic behavior," *International Conference on Computer-Aided Design,* Santa Clara, pp. 131-133, Nov. 12-15, 1984.

[26] P. Banerjee and J. A. Abraham, "Fault Characterization of VLSI circuits," *IEEE International Conference on Circuits and Computers,* New York, pp. 564-568, Sept. 1982.

[27] I. Jansch and B. Courtois, "Design of Checkers based on analytical fault hypotheses," (preliminary report) IMAG Report no. RR 379, Mar. 1983.

[28] M. Nicolaidis and B. Courtois, "Design of self-checking systems based on analytical fault hypotheses," IMAG Report no. RR 353, Mar. 1983.

[29] R. H. Krambeck, C. M. Lee and H. S. Law, "High-speed compact circuits with CMOS," *IEEE Journal of Solid-State Circuits,* Vol. SC-17, No. 3, pp. 614-619, June 1982.

[30] V. G. Oklobdzija and P. G. Kovijanic, "On testabilty of CMOS-Domino logic," *14th International Symposium on Fault-Tolerant Computing,* Orlando, pp. 50-55, June 20-22, 1984.

[31] R. Chandramouli, "On testing stuck-open faults," *13th International Symposium on Fault-Tolerant Computing,* Milan, Italy, pp. 258-265, June 28-30, 1983.

[32] K. W. Chiang and Z. G. Vranesic, "On fault detection in CMOS logic networks," *20th Design Automation Conference,* pp. 50-56, June 1983.

[33] 20th Design Automation Conference, pp. 50-56, June 1983. Y.M. El-Ziq, "Automatic test generation for stuck-open faults in CMOS VLSI," *18th Design Automation Conference,* Nashville, pp. 347-354, June 1981.

[34] Y.M. El-Ziq and R.J. Cloutier, "Functional-level test generation for stuck-open faults in CMOS VLSI," *International Test Conference,* Philadelphia, pp. 536-546, October 1981.

[35] S.K. Jain and V.D. Agrawal, "Test generation for MOS circuits using D-algorithm," *20th Design Automation Conference,* pp. 64-70, June 1983.

[36] S.M. Reddy, V.D. Agrawal and S.K. Jain, "A gate level model for CMOS combinational logic circuits with application to fault detection," *21st Design Automation Conference,* pp. 504-509, June 1984.

[37] S. B. Akers, "Universal test sets for logic networks," *IEEE Transactions on Computers,* Vol. C-22, pp. 1016-1020, Nov. 1973.

[38] R. Betancourt, "Derivation of minimum test sets for unate logical circuits," *IEEE Transactions on Computers,* Vol. C-20, pp. 1264-1269, Nov. 1971.

[39] S.M. Reddy, "Complete test sets for logic functions," *IEEE Transactions on Computers,* Vol. C-22, pp. 1016-1020, Nov. 1973.

[40] S. M. Reddy, "A note on self-checking checkers," *IEEE Transactions on Computers,* Vol. C-23, pp. 1100-1102, Oct. 1974.

[41] C. Efstathiou and C. Halatsis, "Modular realization of totally self-checking checkers for m-out-of-n codes," *13th International Symposium on Fault-Tolerant Computing,* Milan, Italy, pp. 154-161, June 1983.

[42] N. Gaitanis and C. Halatsis, "A new design method for m-out-of-n TSC checkers," *IEEE Transactions on Computers,* Vol. C-32, pp. 273-283, Mar. 1983.

[43] S. Piestrak, "Design method of totally self-checking checkers for m-out-of-n codes," *13th International Symposium on Fault-Tolerant Computing,* Milan, Italy, pp. 162-168, June 1983.

[44] G. P. Mak, "The design of Programmable Logic Arrays with concurrent error detection," Ph.D. thesis, Univ. of Illinois, Urbana, Illinois, 1982.

[45] T. Nanya and Y. Tohma, "A 3-level realization of totally self-checking checkers for m-out-of-n codes," *13th International Symposium on Fault-Tolerant Computing,* Milan, Italy, pp. 173-176, June 1983.

[46] J. Khakbaz, "Self-testing embedded parity trees," *12th International Symposium on Fault-Tolerant Computing,* Santa Monica, pp. 109-116, June 22-24 1982.

[47] S. L. Wang and A. Avizienis, "The design of totally self-checking circuits using Programmable Logic Arrays," *9th International Symposium on Fault-Tolerant Computing,* Madison, Wisconsin, pp. 173-180, June 1979.

[48] C.W. Cha, "A testing strategy for PLAs," *Proceedings 15th Design Automation Conference,* pp. 326-334, June 1978.

[49] J.E. Smith, "Detection of faults in Programmable Logic Arrays," IEEE Computer Society Repository R78-2, Jan. 1978.

[50] D.L. Ostapko and S.J. Hong, "Fault analysis and test generation for Programmable Logic Array," *8th International Symposium on Fault-Tolerant Computing,* pp.83-89, June 1978.

[51] B. Bose and T.R.N. Rao, "Theory of unidirectional error correcting/detecting codes," *IEEE Transactions on Computers,* Vol. C-31, pp. 521-530, June 1982.

[52] J.M. Berger, "A note on error detection codes for asymmetric channels," *Information and Control,* Vol. 4, pp. 68-73, Mar. 1961.

[53] P. Agrawal, "Test generation at the switch level," *International Conference on Computer-Aided Design,* Santa Clara, California, pp. 128-130, Nov. 1984.

[54] S. M. Reddy, M. K. Reddy and V. D. Agrawal, "Robust tests for stuck-open faults in CMOS combinational logic circuits," *International Symposium on Fault-Tolerant Computing,* Orlando, Florida, pp. 44-49, June 1984.

VITA

Niraj Kumar Jha was born in Bihar, India on January 10, 1960. He received a B.Tech. degree in Eletronics and Electrical telecommunications Engineering from I.I.T., Kharagpur, India in 1981, and an M.S. degree in Electrical Engineering from State University of New York at Stonybrook in 1982. He was a Research Assistant with the Computer Systems Group at the Coordinated Science Laboratory, University of Illinois, Urbana from 1982 to 1985.

END

DTIC

8-86